Open Group Technical Standard

DRDA Volume 3:

Distributed Data Management (DDM) Architecture

The Open Group

Open Group Technical Standard

DRDA Volume 3: Distributed Data Management (DDM) Architecture

ISBN: 1-85912-206-X
Document Number: C814

Published in the U.K. by The Open Group, October 1998.

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

List of Figures

List of Tables

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

The ''X'' Device is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical Standards and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

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

- *Preliminary Specifications*

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a Technical Standard. While the intent is to progress Preliminary Specifications to corresponding Technical Standards, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

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

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as Technical Standards, in which case the relevant Technology Specification is superseded by a Technical Standard.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the Technical Standards or Preliminary Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

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

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

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

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/pubs*.

This Document

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

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

This volume, *Distributed 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 Open Group Technical Standard, however, should be sent to the addresses provided on the copyright page.

Trademarks

Motif®, OSF/1®, UNIX®, and the ''X Device''® are registered trademarks and IT DialTone™ and The Open Group™ are trademarks of The Open Group in the U.S. and other countries.

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

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

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

Intel® is a registered trademark of Intel Corporation.

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

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

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

VAX® is a registered trademark of Digital Equipment Corporation.

Referenced Documents

These publications provide the background for understanding DRDA.

DRDA Overview

For an overview of DRDA, read:

- Open Group Technical Standard, TBD 1998, DRDA Volume 1: Distributed Relational Database Architecture (DRDA) (ISBN: 1-85912-295-7, C812).

The DRDA Processing Model and Command Flows

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

- Open Group Technical Standard, TBD 1998, DRDA Volume 3: Distributed Data Management (DDM) Architecture (ISBN: 1-85912-206-X, C814) (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:

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

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

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

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

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

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

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

Data Definition and Exchange

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

- Open Group Technical Standard, TBD 1998, DRDA Volume 2: Formatted Data Object Content Architecture (FD:OCA) (ISBN: 1-85912-201-9, C813).
- ISO/IEC 9075: 1992, Information Technology — Database Language SQL (technically identical to ANSI standard X3.135-1992).
- *Character Data Representation Architecture Reference*, SC09-1390 (IBM).
- *Character Data Representation Architecture Registry*, SC09-1391 (IBM).
- *Character Data Representation Architecture, Executive Overview*, GC09-1392 (IBM).

Other

- *ANSI/IEEE Std. 745-1985, Binary Floating Point Arithmetic*.
- *Dictionary of Computing—Information Processing, Personal Computing, Telecommunications, Print and Office Systems, IBM-Specific Terms*, SC20-1699 (IBM).

Chapter 1

The DRDA Specification

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

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

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

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

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

It is recommended that the DRDA Reference be used as the main source of information and roadmap for implementing DRDA. This section describes the relationships 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 xxii.)

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. For more detailed information, see the *Distributed Data Management Architecture Implementation Programmer's Guide* (SC21-9529, IBM) and the DDM Reference.

Key DDM Concepts

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

Table 1-1 DDM Modeling and Description Terms

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

Key DDM Concepts for DRDA Implementation

After becoming familiar with the DDM overview terms in Table 1-1 on page 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

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

DDM Command Objects in DRDA

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

Table 1-3 DDM Command Objects Used by DRDA

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

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

Chapter 2

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), 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.

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 single system may have one or more servers. A single server can have a collection of files, a relational database, or both. As an example, each of the DB2 subsystem of an IBM OS/390 can be viewed as a server that manages its own collection of data.

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

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 code point 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 code point in the proper dictionary.

The code point 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 and is not addressed by DDM architecture. 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.

2.5.5 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 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 code point 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 code point 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 code point 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 code points in the dictionaries from the DDM Reference. While agreements between products to support extensions are permitted, they must not be required.
- Product extension code points 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 code points to their own product-defined extensions that do not conflict with the code points 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
- Code point
- 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 code point field
- Total length of the object's data

The code point 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 manpage). 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 code point, if any, for the term.

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

   QDDTTRD
   : DDM Tutorial Objects Dictionary (code point: NONE)

   QDDPRMD
   : DDM Primitive Classes Dictionary (code points in X'0000' range)

   QDDBASD
   : DDM Base Classes Dictionary (code points in X'1nnn' range)

   QDDRDBD
   : DDM RDB Classes Dictionary (code points 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 book marks 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.

ABBREVIATIONS

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 3-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 message end in RM, and all access method names end in AM.

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

Abbreviation	Meaning
ABN	abnormal
ABS	absolute
ACC	access
ACT	active
ADR	address
AGN	agent
AI	access intent
AL	attribute list
ALC	allocate
ALT	alternate

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Abbreviation	Meaning
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

Variable	Reference
semantic	*CONCEPTS* on page 237

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

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

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274

Variable	Reference
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	REBIND on page 606
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	PRPSQLSTT on page 533
semantic	*CNTQRY* on page 217
	DSS on page 289
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	QRYNOPRM on page 562

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

Variable	Reference
insvar	*AGNPRMRM* on page 61
	CMDCHKRM on page 170
	RSCLMTRM on page 634
	SVRCOD on page 756

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 *rdbnam* parameter specifies the name of the RDB.

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

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

The *prddta* parameter carries product-specific information.

The *sttdecdel* 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 *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 *trgdftrt* parameter controls whether to return the target default values in ACCRDBRM.

The ACCRDBRM is returned after the RDB is successfully accessed.

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.

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.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2001'	
rdbacccl	INSTANCE_OF ENUVAL REQUIRED CMDTRG	RDBACCCL - RDB Access Manager Class X'2407' - SQLAM - SQL Application Manager
crrtkn	INSTANCE_OF REQUIRED MINLVL	CRRTKN - Correlation Token [1] 4
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
prdid	INSTANCE_OF REQUIRED	PRDID - Product-Specific Identifier
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
rdbalwupd	INSTANCE_OF OPTIONAL	RDBALWUPD - RDB Allow Updates
prddta	INSTANCE_OF OPTIONAL IGNORABLE	PRDDTA - Product-Specific Data

0. This instance variable is OPTIONAL in DDM Level 3.

sttdecdel	INSTANCE_OF OPTIONAL	STTDECDEL - Statement Decimal Delimiter
sttstrdel	INSTANCE_OF OPTIONAL	STTSTRDEL - Statement String Delimiter
trgdftrt	INSTANCE_OF OPTIONAL	TRGDFTRT - Target Default Value Return
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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
X'0035'	INSTANCE_OF OPTIONAL MTLINC	TYPDEFOVR - TYPDEF Overrides X'2408' - SQLCARD - SQL Communications Area Reply Data
X'2408'	INSTANCE_OF OPTIONAL NOTE MTLINC MTLINC	SQLCARD - SQL Communications Area Reply Data Returned if and only if RDBAFLRM is returned. An SQLCARD always follows the RDBAFLRM. X'002F' - TYPDEFNAM - Data Type Definition Name X'0035' - TYPDEFOVR - TYPDEF Overrides
cmdrpy		**COMMAND REPLIES**
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	RDBAFLRM - RDB Access Failed Reply Message
	NOTE	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

Variable	Reference
clscmd	*SQLAM* on page 694
cmddta	*BNDSQLSTT* on page 130
	DSCRDBTBL on page 281
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
insvar	*ACCRDBRM* on page 48
rpydta	*CNTQRY* on page 217
semantic	*ACCRDBRM* on page 48
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DFTPKG on page 267
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRDDTA on page 528

Variable	Reference
	PRPSQLSTT on page 533
	RDBACCRM on page 578
	RDBCMM on page 582
	RDBNACRM on page 587
	RDBOVR on page 593
	RDBRLLBCK on page 598
	REBIND on page 606
	RSCLMTRM on page 634
	SQL on page 680
	SQLAM on page 694
	SRVLST on page 725
	SYNCPTOV on page 787
	TRGDFTRT on page 867
	TYPDEFOVR on page 876

ACCRDBRM

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 *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 *prdid* 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.

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 *pkgdftcst* 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. This parameter is returned if the value of TRGDFTRT is TRUE in ACCRDB.

The *usrid* parameter specifies the target-defined user ID. This parameter is returned if the value of TRGDFTRT is TRUE in ACCRDB.

The *srvlst* 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.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2201'	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code 4 - WARNING - Warning Severity Code

prdid	INSTANCE_OF REQUIRED	PRDID - Product-Specific Identifier
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
rdbinttkn	INSTANCE_OF OPTIONAL	RDBINTTKN
crrtkn	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.
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
pkgdftcst	INSTANCE_OF ENUVAL ENUVAL ENUVAL OPTIONAL NOTE	PKGDFTCST - Package Default Character Subtype 'CSTBITS' 'CSTSBCS' 'CSTMBCS' This parameter is returned if the value of TRGDFTRT (TRUE) is specified in ACCRDB.
usrid	INSTANCE_OF OPTIONAL NOTE	USRID - User ID at the Target System This parameter is returned if the value of TRGDFTRT (TRUE) is specified in ACCRDB.
srvlst	INSTANCE_OF OPTIONAL MINLVL	SRVLST - Server List 5
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
rpydta	*BGNBND* on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
semantic	*ACCRDB* on page 42
	SRVLST on page 725
	SYNCPTOV on page 787
	TRGDFTRT on page 867

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 undergo 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 and it is identical to the SECMEC value in the ACCSEC command. If the target server does not support the SECMEC requested by the source server, then one or more values in SECMEC are returned and the source server must choose one of these values for the security mechanism. The selected value is presented in the SECCHK 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 undergo conversion between the code pages.

Exceptions

If the ACCSEC command is command 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 with the SECCHK command. If the source server does not support any of the security mechanism 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, then it continues processing with a SECCHK command containing the selected SECMEC value and the SECTKN value for the selected security mechanism.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'106D'	
secmgrnm	INSTANCE_OF OPTIONAL IGNORABLE NOTE CMDTRG	SECMGRNM - Security Manager Name This parameter has a null value and does not have to be validated.
secmec	INSTANCE_OF REQUIRED	SECMEC - Security Mechanism
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.
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**

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.
cmdrpy		**COMMAND REPLIES**
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'2211'	INSTANCE_OF NOTE	RDBNFNRM - RDB Not Found If the specified rdbnam is not found, or if the target server requires the presence of an rdbnam but the rdbnam 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

Variable	Reference
inscmd	*SECMGR* on page 663
insvar	*ACCSECRD* on page 55
	PRCCNVCD on page 521
semantic	*ACCSECRD* on page 55
	DCESEC on page 247
	DCESECOVR on page 248
	SECCHK on page 652
	SECMGR on page 663
	SYNCPTOV on page 787
	TCPCMNI on page 840
	USRIDNWPWD on page 889
	USRIDONL on page 890

Variable	Reference
	USRIDPWD on page 891
	USRSECOVR on page 893

NAME

ACCSECRD — Access Security Reply Data

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'14AC'	
secmec	INSTANCE_OF NOTE REQUIRED	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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SECMEC* on page 661
rpydta	*ACCSEC* on page 51
semantic	*ACCSEC* on page 51
	DCESECOVR on page 248
	SYNCPTOV on page 787
	TCPCMNI on page 840
	USRSECOVR on page 893

AGENT

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 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 663 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 code point 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 one. See the description of the term CCSIDMGR. 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 of ASCII 819 that support the CCSIDMGR can send character data in the DDM commands and reply messages in that code page and not need 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.

Agent Processing

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

AGNCMDPR
: Target Agent Command Processing (*AGNCMDPR* on page 60)

AGNDCLPR
: Target Agent Declare Name Processing

AGNRPYPR
: Source Agent Reply Message Processing (*AGNRPYPR* on page 63)

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

DDM Levels	**1**	**2**	**3**	**4**	**5**
AGENT *Manager Dependencies*	1	2	3	4	5
DICTIONARY	1	1	1	1	1
SECMGR[3]	1	1	1	1	4
CCSIDMGR				4	4
RSYNCMGR				5	5
SYNCPTMGR					5
Managers Routed-to					
SUPERVISOR	1	1	1	1	1
SQLAM			3	4	5
RSYNCMGR				5	5
SYNCPTMGR					5
Communications Managers					
CMNAPPC	1	1	3	3	3
CMNSYNCPT				4	4
CMNTCPIP					5

mgrlvln	5	
mgrdepls		**MANAGER DEPENDENCY LIST**
X'1458'	INSTANCE_OF	DICTIONARY - Dictionary
	NOTE	Dictionaries resolve code points to class descriptors for validation and routing functions.
	MGRLVLN	1
X'1440'	INSTANCE_OF	SECMGR - Security Manager
	NOTE	The Security Manager validates the authorizations of the requester to specific commands and dictionary classes.
	MGRLVLN	5
	NOTE	The SECMGR can be at DDM Level 1 or DDM Level 5.
	MINLVL	5
X'14CC'	INSTANCE_OF	CCSIDMGR - CCSID Manager
	NOTE	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.
	MGRLVLN	4
	MINLVL	4
	OPTIONAL	
vldattls		**VALID ATTRIBUTES**
X'1152'	INSTANCE_OF	AGNNAM - Agent Name
X'0019'	INSTANCE_OF	HELP - Help Text
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
mgrdepls	*SQLAM* on page 694
semantic	*EXTENSIONS* on page 346
	INHERITANCE on page 380
	MANAGER on page 417
	RDBOVR on page 593
	SQLAM on page 694

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

Variable	Reference
	SUBSETS on page 747
	SYNCPTMGR on page 782

AGNCMDPR

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

Variable	Reference
semantic	*AGENT* on page 56
	CMDTRG on page 175
	COMMAND on page 233
	SUBSETS on page 747

NAME

AGNPRMRM — Permanent Agent Error

AGNPRMRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘1232’	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL ENUVAL	SVRCOD - Severity Code 16 - SEVERE - Severe Error Severity Code 32 - ACCDMG - Access Damage Severity Code 64 - PRMDMG - Permanent Damage Severity Code
reccnt	INSTANCE_OF MINVAL OPTIONAL 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.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
semantic	*DSS* on page 289
	TCPTRGER on page 861

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:[4]
    1. ORDERED—The parameters must be sent in the order that the DDM Architecture defines.
2. Large Objects
    1. 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 code point of the object and its envelopes.

SEE ALSO

Variable	Reference
semantic	*AGENT* on page 56

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

APPCMNFL

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 65.

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).
- The SCM and the TCM both perform synchronous sends and receives.

```
Source Communications Mgr.                       Target Communications Mgr.

   .                                                .
   .                                                .
send state                                       receive state

SEND_DATA                          (1)           RECEIVE_AND_WAIT
   RESOURCE(res_var)                                RESOURCE(res_var)
   DATA(cmd_var)                                    FILL(LL)
   LENGTH(lng_var)                                  LENGTH(lng_var)
   RETURN_CODE(rc_var)                              RETURN_CODE(rc_var)
   REQUEST_TO_SEND_RECEIVED      FAILURE            REQUEST_TO_SEND_RECEIVED
      (rts_var)                    (2)                 (rts_var)
      :                                             DATA(dta_var)
      :                                             WHAT_RECEIVED(typ_var)
   rc_var =                   (3)                   :
   RESOURCE_FAILURE_ENTRY            (4)            lng_var = (unchanged)
      rts_var = NO                                  rts_var = NO
      (cleanup)               (5)                   rc_var =
                                                       RESOURCE_FAILURE_RETRY
DEALLOCATE                    (7)                   typ_var = (not set)
   RESOURCE(res_var)                 (6)               (cleanup)
   TYPE(LOCAL)
   RETURN_CODE(rc_var)               (8)         DEALLOCATE
   :                                                RESOURCE(res_var)
   rc_var = OK                                      TYPE(LOCAL)
                                                    RETURN_CODE(rc_var)
                                                    :
                                                    rc_var = OK
```

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

Variable	Reference
semantic	*APPCMNT* on page 72
	CMNAPPC on page 179

NAME

APPCMNI — LU 6.2 Communications Initiation

APPCMNI

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 69. 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 69.

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.

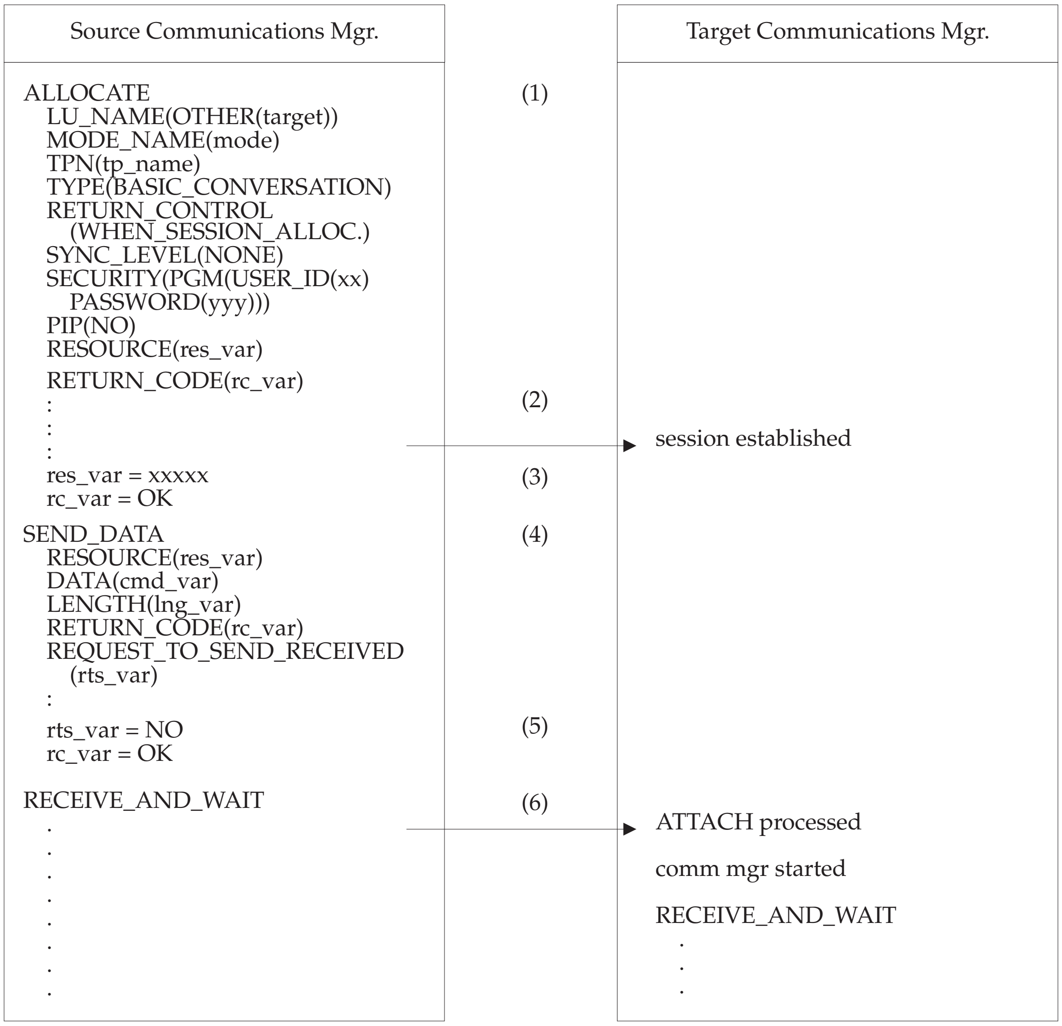


Figure 3-2 Communications Initiation by Source System

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 the term CMNAPPC).

  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

Variable	Reference
semantic	*APPCMNFL* on page 64
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CMNAPPC on page 179

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 the term APPCMNFL 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 the term APPCMNI).
- No error situation occurs.

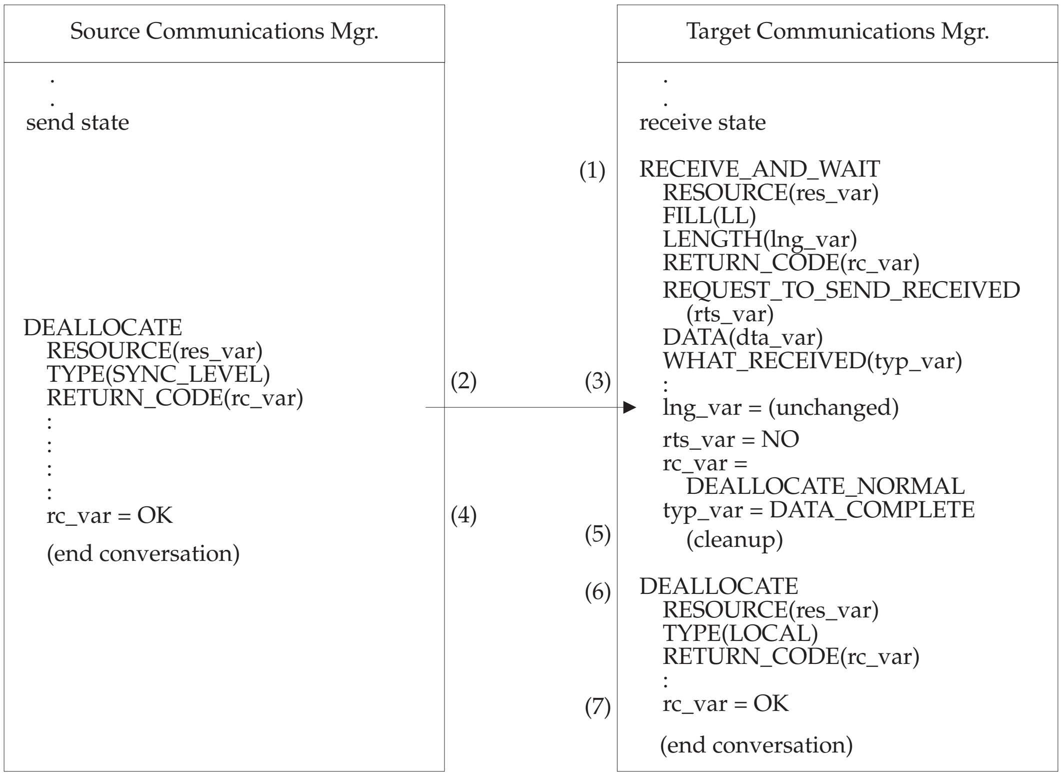


Figure 3-3 Normal Communications Termination

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
- A 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

Variable	Reference
semantic	*CMNAPPC* on page 179

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 76. 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 the terms APPCMNI and SYNCMNI).
- The response to the RQSDSS is a single RPYDSS.
- No error situations occur.

```
Source Communications Mgr.                                Target Communications Mgr.

  .                                                         .
  .                                                         .
  .                                                         .
(send state)                                              (receive state)

SEND_DATA                          (2)          (1)       RECEIVE_AND_WAIT
  RESOURCE(res_var)                                         RESOURCE(res_var)
  DATA(cmd_var)                                             FILL(LL)
  LENGTH(lng_var)                                           LENGTH(lng_var)
  RETURN_CODE(rc_var)                                       RETURN_CODE(rc_var)
  REQUEST_TO_SEND_RECEIVED                                  REQUEST_TO_SEND_RECEIVED
    (rts_var)                                                 (rts_var)
  :                                                         DATA(dta_var)
  rc_var = OK                      (3)                      WHAT_RECEIVED(type_var)
  rts_var = NO                                              :
                                                            :
SEND_DATA                          (4)                      :
  RESOURCE(res_var)                                         :
  DATA(dta_var)                                             :
  LENGTH(lng_var)                                           :
  RETURN_CODE(rc_var)                                       :
  REQUEST_TO_SEND_RECEIVED                                  :
    (rts_var)                                               :
  :                                                         :
  rc_var = OK                      (5)                      :
  rts_var = NO                                              :
                                                            :
RECEIVE_AND_WAIT                   (6)          (7)         lng_var = command length
  RESOURCE(res_var)            ------------------------->   rc_var = OK
  FILL(LL)                                                  rts_var = NO
  LENGTH(lng_var)                                           dta_var = DDM command
  RETURN_CODE(rc_var)                                       type_var = DATA_COMPLETE
  REQUEST_TO_SEND_RECEIVED
    (rts_var)
  DATA(buf_var)
  WHAT_RECEIVED(type_var)
  :
```

Figure 3-4 Source Sends Command with Data (APPSRCCD) Protocol (Part 1)

```
(receive state)
:
:

(8)  RECEIVE_AND_WAIT
       RESOURCE(res_var)
       FILL(LL)
       LENGTH(lng_var)
       RETURN_CODE(rc_var)
       REQUEST_TO_SEND_RECEIVED
         (rts_var)
       DATA(dta_var)
       WHAT_RECEIVED(type_var)
       :
(9)    lng_var = cmd data lng
       rc_var = OK
       rts_var = NO
       dta_var = DDM cmd data
       type_var = DATA_COMPLETE
(10) RECEIVE_AND_WAIT
       RESOURCE(res_var)
       FILL(LL)
       LENGTH(lng_var)
       RETURN_CODE(rc_var)
       REQUEST_TO_SEND_RECEIVED
         (rts_var)
       DATA(dta_var)
       WHAT_RECEIVED(type_var)
       :
       :
(11)   lng_var = (unchanged)
       rc_var = OK
       rts_var = NO
       type_var = SEND

     (send state)

(12) SEND_DATA
       RESOURCE(res_var)
       DATA(rpy_var)
       LENGTH(lng_var)
       RETURN_CODE(rc_var)
       REQUEST_TO_SEND_RECEIVED
         (rts_var)
       :
       :
(13)   rc_var = OK
       rts_var = NO
```

Figure 3-5 Source Sends Command with Data (APPSRCCD) Protocol (Part 2)

```
  :                                        (15)   (14)   RECEIVE_AND_WAIT
  :                                                        .
  lng_var = reply length          <--------------------    .
  rc_var = OK                                              .
  rts_var = NO
  dta_var = reply data                                   (receive state)
  type_var = DATA_COMPLETE

RECEIVE_AND_WAIT                           (16)
  RESOURCE(res_var)
  FILL(LL)
  LENGTH(lng_var)
  RETURN_CODE(rc_var)
  REQUEST_TO_SEND_RECEIVED
    (rts_var)
  DATA(buf_var)
  WHAT_RECEIVED(type_var)
  :
  lng_var = (unchanged)                    (17)
  rc_var = OK
  rts_var = NO
  type_var = SEND

(send state)
  .
  .
  .
```

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 the term APPCMNI). 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

Variable	Reference
semantic	*CMNAPPC* on page 179
	CMNSYNCPT on page 197

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 83. 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 the terms APPCMNI and SYNCMNI).
- No error situations occur.

```
Source Communications Mgr.                       Target Communications Mgr.

   .                                                .
   .                                                .
   .                                                .
(send state)                                     (receive state)

SEND_DATA                          (2)      (1)  RECEIVE_AND_WAIT
   RESOURCE(res_var)                                RESOURCE(res_var)
   DATA(cmd_var)                                    FILL(LL)
   LENGTH(lng_var)                                  LENGTH(lng_var)
   RETURN_CODE(rc_var)                              RETURN_CODE(rc_var)
   REQUEST_TO_SEND_RECEIVED                         REQUEST_TO_SEND_RECEIVED
      (rts_var)                                        (rts_var)
   :                                                DATA(dta_var)
   rc_var = OK                     (3)              WHAT_RECEIVED(type_var)
   rts_var = NO                                     :
                                                    :
                                                    :
RECEIVE_AND_WAIT                                    :
   RESOURCE(res_var)               (4)      (5)     :
   FILL(LL)                  ---------------------> lng_var = command length
   LENGTH(lng_var)                                  rc_var = OK
   RETURN_CODE(rc_var)                              rts_var = NO
   REQUEST_TO_SEND_RECEIVED                         dta_var = DDM command
      (rts_var)                                     type_var = DATA_COMPLETE
   DATA(buf_var)
   WHAT_RECEIVED(type_var)                  (6)  RECEIVE_AND_WAIT
   :                                                RESOURCE(res_var)
   :  (receive state)                               FILL(LL)
   :                                                LENGTH(lng_var)
   :                                                RETURN_CODE(rc_var)
   :                                                REQUEST_TO_SEND_RECEIVED
   :                                                   (rts_var)
   :                                                DATA(dta_var)
   :                                                WHAT_RECEIVED(type_var)
   :                                                :
   :                                                :
   :                                        (7)     lng_var = (unchanged)
   :                                                rc_var = OK
   :                                                rts_var = NO
   :                                                type_var = SEND
```

Figure 3-7 Source Sends Command with No Command Data (APPSRCCR) Protocol (Part 1)

: : :			(send state)
: : : : : : : :		(8)	SEND_DATA RESOURCE(res_var) DATA(rpy_var) LENGTH(lng_var) RETURN_CODE(rc_var) REQUEST_TO_SEND_RECEIVED (rts_var) : :
: :		(9)	rc_var = OK rts_var = NO
lng_var = reply length rc_var = OK rts_var = NO dta_var = reply data type_var = DATA_COMPLETE	(11)	(10)	RECEIVE_AND_WAIT . . . (receive state)
RECEIVE_AND_WAIT RESOURCE(res_var) FILL(LL) LENGTH(lng_var) RETURN_CODE(rc_var) REQUEST_TO_SEND_RECEIVED (rts_var) DATA(buf_var) WHAT_RECEIVED(type_var) :	(12)		
lng_var = (unchanged) rc_var = OK rts_var = NO type_var = SEND	(13)		
(send state)			

Figure 3-8 Source Sends Command with No Command 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 the term APPCMNI). 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

Variable	Reference
semantic	*APPTRGER* on page 91
	CMNAPPC on page 179
	CMNSYNCPT on page 197

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 88. 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 the terms APPCMNI and SYNCMNI).

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.

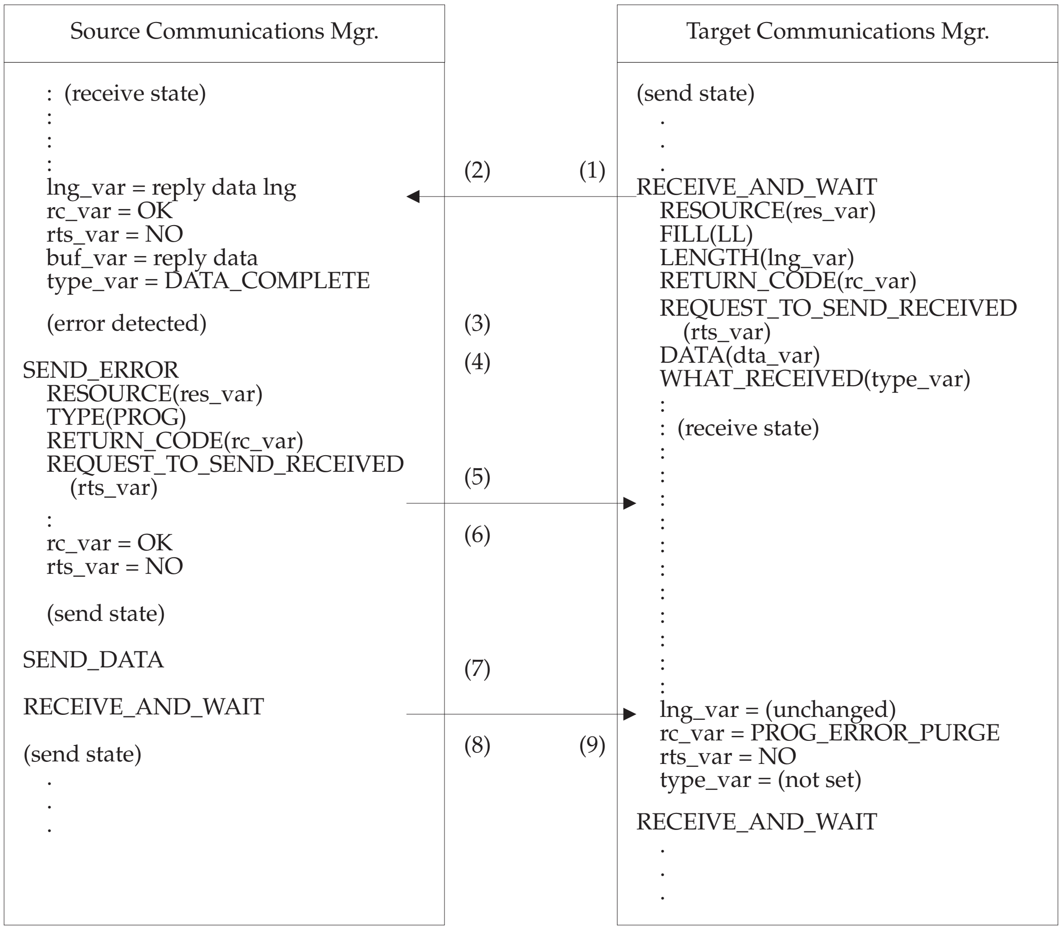


Figure 3-9 Source System Detects Error (APPSRCER) Protocol

Figure Notes

(1) 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.

(2) The TCM SNA LU 6.2 communications facility sends all of the buffered data to the source system. Once the source SNA LU 6.2 communications facility receives the data, the SCM RECEIVE_AND_WAIT verb completes its operation, and control is returned to 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

Variable	Reference
semantic	*CMNAPPC* on page 179
	CMNSYNCPT on page 197

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 92. 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 the terms APPCMNI and SYNCMNI).

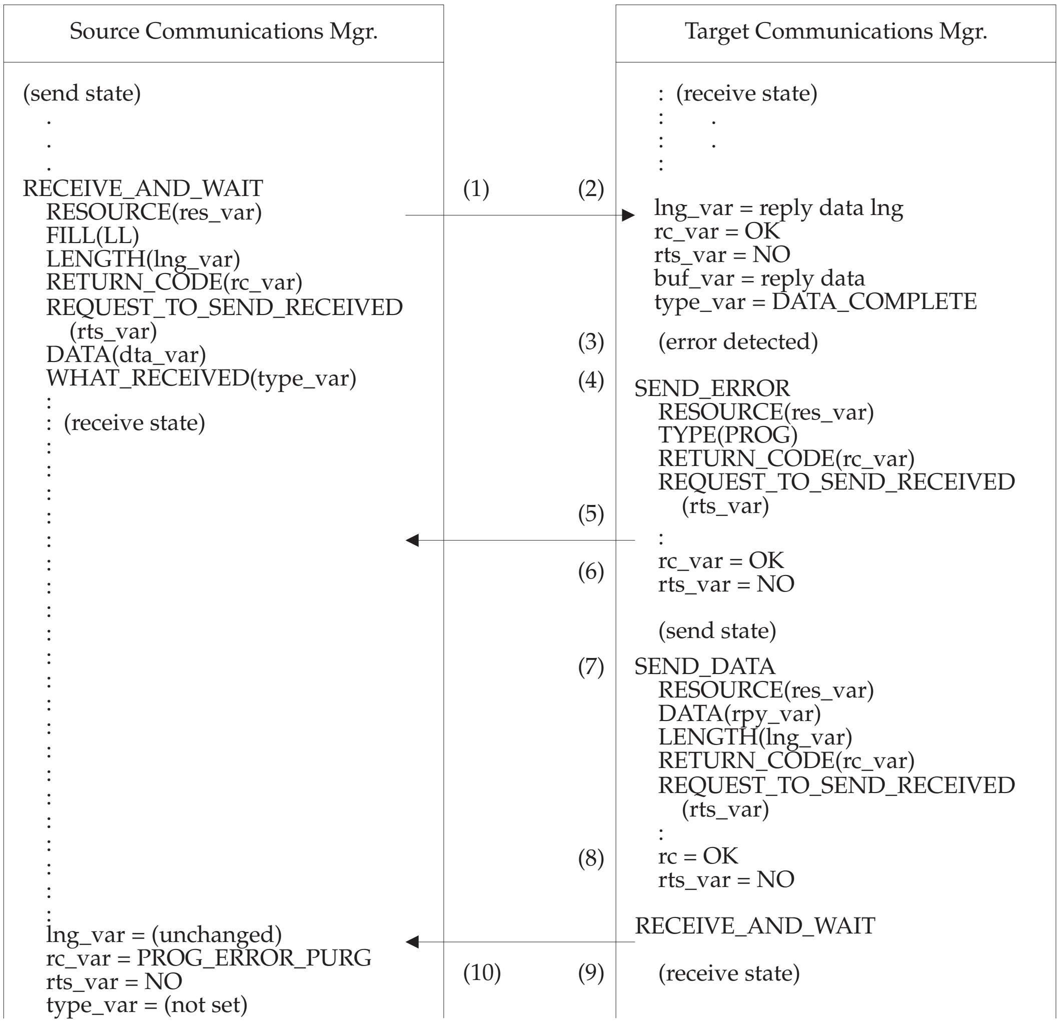


Figure 3-10 Target System Detects Error (APPTRGER) Protocol (Part 1)

```
RECEIVE_AND_WAIT                         (11)
   RESOURCE(res_var)
   FILL(LL)
   LENGTH(lng_var)
   RETURN_CODE(rc_var)
   REQUEST_TO_SEND_RECEIVED
      (rts_var)
   DATA(dta_var)
   WHAT_RECEIVED(type_var)
   :
   lng_var = length of repl              (12)
   rc_var = OK
   rts_var = NO
   type_var =DATA_COMPLETE
RECEIVE_AND_WAIT                         (13)
   RESOURCE(res_var)
   FILL(LL)
   LENGTH(lng_var)
   RETURN_CODE(rc_var)
   REQUEST_TO_SEND_RECEIVED
      (rts_var)
   DATA(buf_var)
   WHAT_RECEIVED(type_var)
   :
   lng_var = (unchanged)                 (14)
   rc_var = OK
   rts_var = NO
   type_var =SEND

   (send state)
```

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

Variable	Reference
semantic	*CMNAPPC* on page 179
	CMNSYNCPT on page 197

ARRAY

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 ')
```

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	

class	X'004B'	
value	REPEATABLE	
	SPRCLS	OBJECT - Self-Identifying Data
	NOTE	An array can have zero or more elements.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*AGENT* on page 56
	SUPERVISOR on page 753

ASSOCIATION

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0001'	
key	SPRCLS	OBJECT - Self-Identifying Data
value	SPRCLS	OBJECT - Self-Identifying Data
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*INHERITANCE* on page 380
sprcls	*DCTINDEN* on page 254
	DEFINITION on page 262
	QLFATT on page 554

NAME

ATTLST — Attribute List

DESCRIPTION (Semantic)

Dictionary	QDDPRMD
Codepoint	X'0046'
Length	*
Class	CLASS
Sprcls	COLLECTION - Collection Object

Attribute List (ATTLST) 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 code point (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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0046'	
attribute	SPRCLS	OBJECT - Self-Identifying Data
	REPEATABLE	
	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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DEFINITION* on page 262
	QLFATT on page 554
semantic	*DEFLST* on page 263
	QLFATT on page 554

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 *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 *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 *vrsnam* 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.

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

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 *bndexpopt* 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 *dgrioprl* 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 *pkgdftcc* 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 *pkgdftcst* 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 *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 *qryblkctl* parameter specifies the query blocking protocol used by all of the queries in the package.

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

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.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2002'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamct	INSTANCE_OF REQUIRED	PKGNAMCT - RDB Package Name and Consistency Token
vrsnam	INSTANCE_OF OPTIONAL	VRSNAM - Version Name
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 *vrsnam* parameter on this command assumes. This parameter is ignored when PKGRPLOPT(PKGRPLNA) is specified.

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
dgrioprl	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	DGRIOPRL - Degree of I/O Parallelism 4
pkgathopt	INSTANCE_OF OPTIONAL	PKGATHOPT - Package Authorization Option
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
pkgrplopt	INSTANCE_OF OPTIONAL	PKGRPLOPT - Package Replacement Option
pkgownid	INSTANCE_OF OPTIONAL	PKGOWNID - Package Owner Identifier
qryblkctl	INSTANCE_OF OPTIONAL	QRYBLKCTL - Query Block Protocol Control
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
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'2405'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	BNDOPT - Bind Option 5
rpydta		**REPLY OBJECTS**
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'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.
cmdrpy		**COMMAND REPLIES**

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

Variable	Reference
inscmd	*SQLAM* on page 694
insvar	*STTDECDEL* on page 742
	STTSTRDEL on page 744
semantic	*BNDCHKEXS* on page 119
	BNDCRTCTL on page 121
	BNDEXSRQR on page 125
	BNDSQLSTT on page 130
	DFTPKG on page 267
	ENDBND on page 307
	FRCFIXROW on page 370
	OPNQRY on page 475
	PKGATHKP on page 492
	PKGATHOPT on page 493
	PKGATHRVK on page 496
	PKGBNARM on page 497

Variable	Reference
	PKGBPARM on page 498
	PKGOWNID on page 512
	PKGRPLALW on page 514
	PKGRPLNA on page 515
	PKGRPLVRS on page 518
	QRYBLKCTL on page 557
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SQL on page 680
	SQLAM on page 694

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

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

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
rpydta	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	PKGRPLNA on page 515

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.

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

SEE ALSO

Variable	Reference
clsvar	*MANAGER* on page 417
insvar	*DCTINDEN* on page 254
semantic	*ABBREVIATIONS* on page 30
	INHERITANCE on page 380
	OBJOVR on page 464
sprcls	*DECPRC* on page 261
	DGRIOPRL on page 270
	ENULEN on page 317
	FDODSCOFF on page 356
	FDODTAOFF on page 359
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362
	LENGTH on page 398
	MAXBLKEXT on page 420
	MAXLEN on page 422

Variable	Reference
	MAXRSLCNT on page 423
	MAXSCTNBR on page 424
	MGRLVLN on page 430
	MINLEN on page 439
	MINLVL on page 440
	NBRROW on page 449
	PKGATHRUL on page 494
	QRYBLKSZ on page 559
	QRYROWNBR on page 569
	SQLSTTNBR on page 715
	SVRCOD on page 756

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.

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

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
insvar	*BIN* on page 110
	DECPRC on page 261
	DGRIOPRL on page 270
	ENULEN on page 317
	FDODSCOFF on page 356
	FDODTAOFF on page 359
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362
	IPADDR on page 388
	LENGTH on page 398
	MAXBLKEXT on page 420
	MAXLEN on page 422
	MAXRSLCNT on page 423
	MAXSCTNBR on page 424
	MGRLVLN on page 430
	MINLEN on page 439
	MINLVL on page 440
	NBRROW on page 449
	OBJDSS on page 455
	OBJECT on page 459
	PKGATHRUL on page 494
	PKGSN on page 519
	QRYROWNBR on page 569
	RPYDSS on page 620
	RQSCRR on page 626
	RQSDSS on page 629
	SECMEC on page 661
	SQLSTTNBR on page 715
	SRVLCNT on page 722
	SRVPRTY on page 730
	SVCERRNO on page 755
	SVRCOD on page 756
	UOWID on page 883

Variable	Reference
semantic	*ATTLST* on page 99
	BIN on page 110
	DTAOVR on page 305
	INHERITANCE on page 380

BITDR

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	ENUVAL ENUVAL	b'0' b'1'
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BITSTRDR* on page 117
	CNSVAL on page 216
	DFTVAL on page 269
	DSSFMT on page 301
	ENUVAL on page 318
	MAXVAL on page 425
	MINVAL on page 441
	RESERVED on page 617
	RSLSETFLG on page 639
	SPCVAL on page 676

Variable	Reference
semantic	*INHERITANCE* on page 380
	LENGTH on page 398
	MAXLEN on page 422
	MINLEN on page 439

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF REPEATABLE	BITDR - A Single Bit
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BINDR* on page 112
	BYTDR on page 137
	CNSVAL on page 216
	DFTVAL on page 269
	ENUVAL on page 318
	HEXDR on page 374
	MAXVAL on page 425
	MINVAL on page 441
	RESERVED on page 617
	SPCVAL on page 676

Variable	Reference
semantic	*INHERITANCE* on page 380
	LENGTH on page 398
	MAXLEN on page 422
	MINLEN on page 439
sprcls	*RSLSETFLG* on page 639

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'211B'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'241D' - BNDEXSOPT - Bind Object Existence Option
	ENUVAL	X'241C' - BNDEXSRQR - Bind Object Existence Required
	DFTVAL	X'241D' - BNDEXSOPT - Bind Object Existence Option
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606

BNDCHKONL

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

Variable	Reference
insvar	*BNDCRTCTL* on page 121
semantic	*BNDNERALW* on page 126

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'211D'	
value	INSTANCE_OF	CODPNTDR - Code Point 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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	BNDCHKEXS on page 119
	BNDEXSRQR on page 125
	BNDNERALW on page 126
	ENDBND on page 307
	PKGOWNID on page 512
	PKGRPLALW on page 514

BNDERRALW

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

Variable	Reference
insvar	*BNDCRTCTL* on page 121
semantic	*ENDBND* on page 307

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 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2130'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'243B' - EXPALL - Explain All Explainable SQL Statements
	ENUVAL	X'243A' - EXPNON - Explain No SQL Statements
	DFTVAL	X'243A' - EXPNON - Explain No SQL Statements
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606

BNDEXSOPT

NAME

BNDEXSOPT — Bind Object Existence Option

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'241D'

Length *

Class codpnt

Bind Object Existence Option (BNDEXSOPT) 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

Variable	Reference
insvar	*BNDCHKEXS* on page 119
semantic	*ENDBND* on page 307

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

Variable	Reference
insvar	*BNDCHKEXS* on page 119

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

Variable	Reference
insvar	*BNDCRTCTL* on page 121

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.

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

SEE ALSO

Variable	Reference
cmddta	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	REBIND on page 606

BNDOPTNM

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
value	INSTANCE_OF MINLEN MAXLEN	CHRSTRDR - Character String 1 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BNDOPT* on page 127
semantic	*BNDOPT* on page 127
	BNDOPTVL on page 129

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...0xFF(valuen>`

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
value	INSTANCE_OF MINLEN MAXLEN	CHRSTRDR - Character String 0 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BNDOPT* on page 127
semantic	*BNDOPT* on page 127

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

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

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 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2004'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
sqlsttnbr	INSTANCE_OF OPTIONAL	SQLSTTNBR - SQL Statement Number
bndsttasm	INSTANCE_OF OPTIONAL	BNDSTTASM - Bind SQL Statement Assumptions
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE REPEATABLE	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 REPEATABLE	TYPDEFOVR - TYPDEF Overrides '' The default means the value received on ACCRDB command is used.

X'2414'	INSTANCE_OF REQUIRED NOTE	SQLSTT - SQL Statement This must precede the SQLSTTVRB command data object.
X'2419'	INSTANCE_OF OPTIONAL NOTE	SQLSTTVRB - SQL Statement Variable Descriptions 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.
rpydta		**REPLY OBJECTS**
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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
insvar	*MAXSCTNBR* on page 424
semantic	*BGNBND* on page 101
	BNDNERALW on page 126
	ENDBND on page 307
	PKGBNARM on page 497
	REBIND on page 606
	SQL on page 680
	SQLAM on page 694
	SQLERRRM on page 710

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.

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

SEE ALSO

Variable	Reference
insvar	*BNDSQLSTT* on page 130
semantic	*BNDSQLSTT* on page 130

BOOLEAN

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.

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

SEE ALSO

Variable	Reference
semantic	*INHERITANCE* on page 380
sprcls	*OUTEXP* on page 489
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	RDBALWUPD on page 580
	RDBCMTOK on page 585
	RLSCONV on page 619
	RTNSQLDA on page 648
	SQLCSRHLD on page 706
	TRGDFTRT on page 867

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'.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF LENGTH	BITSTRDR - Bit String Field 8
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RSYNCTYP* on page 647
	SYNCTYPE on page 830
	UOWSTATE on page 885
semantic	*INHERITANCE* on page 380

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'119D'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGDFTCC* on page 501
	TYPDEFOVR on page 876
semantic	*ENDBND* on page 307
	TYPDEFOVR on page 876

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'119E'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGDFTCC* on page 501
	TYPDEFOVR on page 876
semantic	*TYPDEFOVR* on page 876

NAME

CCSIDMGR — CCSID Manager

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint 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 code points.

The initial EXCSAT command negotiates the desired encoding scheme. The source server in the *mgrlvlls* parameter includes the code point of the CCSIDMGR and the CCSID of the code page it uses to send character data. The target server in the *mgrlvlls* parameter returns the code point 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 *mgrlvlls* 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 *mgrlvlls* parameter contains the code point 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 *mgrlvlls* parameter returned for the EXCSAT command contains the code point 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 *mgrlvlls* parameter, then the target server returns the value FFFF in the *mgrlvlls* 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
```

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'14CC'	
clscmd	**NIL**	

inscmd	**NIL**	
mgrlvln	REDEFINED	
mgrlvln	INSTANCE_OF NOTE SPCVAL NOTE SPCVAL NOTE	CCSID - Coded Character Set Identifier The CCSID is an UNSBINDR number. CCSIDs 500, 819, and 850 must be supported. Other CCSIDs are optional. 0 When sent by the target server, zero means the CCSIDMGR is not supported. 65535 When sent by the target server, this value means the CCSID the source server requested is not supported. 65535 is FFFF.
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
mgrdepls	*AGENT* on page 56
semantic	*ACCRDB* on page 42
	ACCSEC on page 51
	AGENT on page 56
	BGNBND on page 101
	BNDOPTNM on page 128
	BNDOPTVL on page 129
	BNDSQLSTT on page 130
	CHRDR on page 145
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331

Variable	Reference
	EXCSQLSTT on page 336
	EXTENSIONS on page 346
	MGROVR on page 436
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652

CCSIDSBC

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'119C'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGDFTCC* on page 501
	TYPDEFOVR on page 876
semantic	*ACCRDB* on page 42
	ENDBND on page 307
	TYPDEFOVR on page 876

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 the description of the term CCSIDMGR.

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

Variable	Reference
insvar	*CHRSTRDR* on page 146
	NAMSYMDR on page 447
semantic	*DTAOVR* on page 305
	INHERITANCE on page 380
	LENGTH on page 398
	MAXLEN on page 422
	MINLEN on page 439

CHRSTRDR

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, abcde 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...'.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF REPEATABLE	CHRDR - A Graphic Character
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
insvar	*BNDOPTNM* on page 128
	BNDOPTVL on page 129
	LOGTSTMP on page 411
	NAMDR on page 444
	NAME on page 445
	NEWPASSWORD on page 450
	NOTE on page 452
	PASSWORD on page 491

Variable	Reference
	RSCNAM on page 637
	SNAADDR on page 674
	SRVCLSNM on page 717
	TEXT on page 864
	TITLE on page 865
	UOWID on page 883
	USRID on page 888
semantic	*INHERITANCE* on page 380
	LENGTH on page 398
	MAXLEN on page 422
	MINLEN on page 439

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 149. See descriptions of the terms INHERITANCE, INHERITED, and SPRCLS. The superclass hierarchy defines a superclass chain from the class's immediate superclass to DATA.

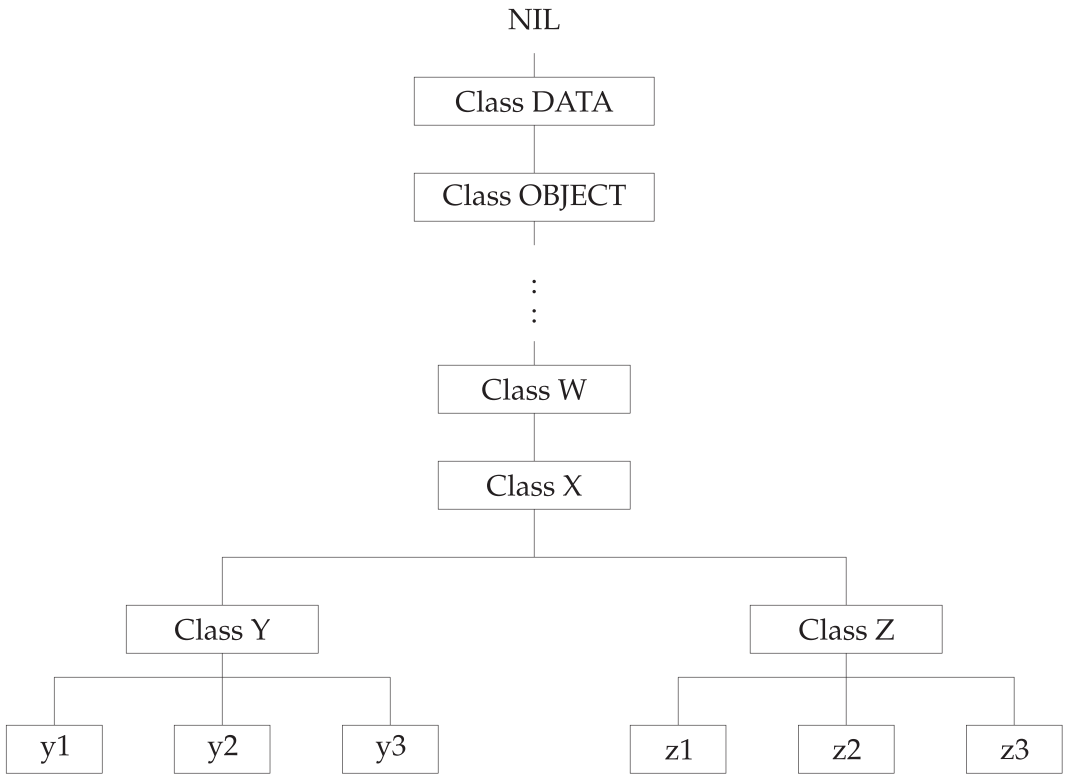


Figure 3-12 Superclass, Class, and Instance Relationships

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 150.

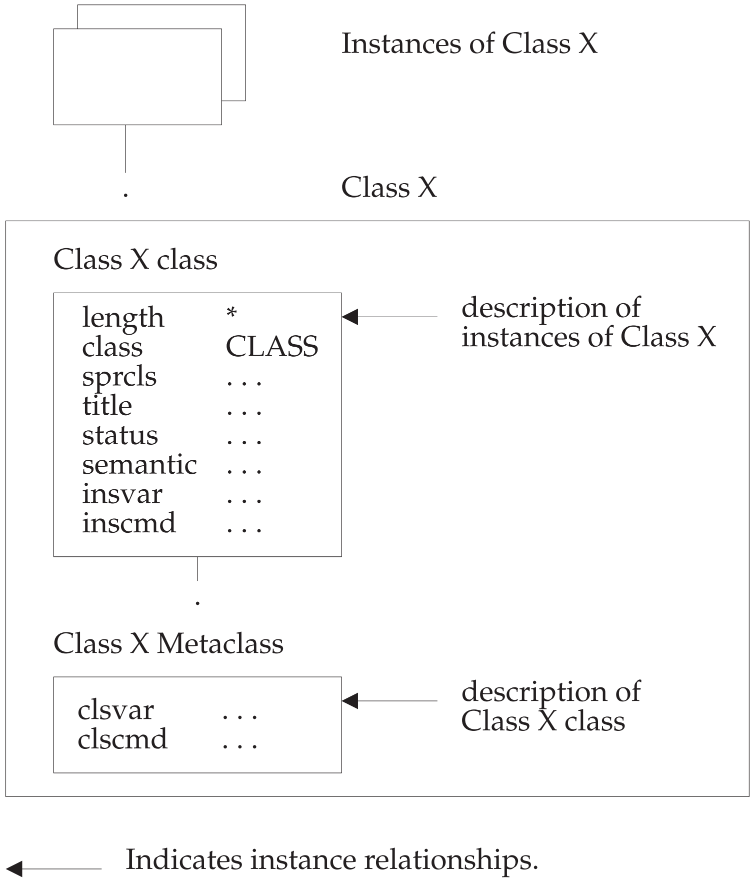


Figure 3-13 Class to Metaclass Relationships

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 151.

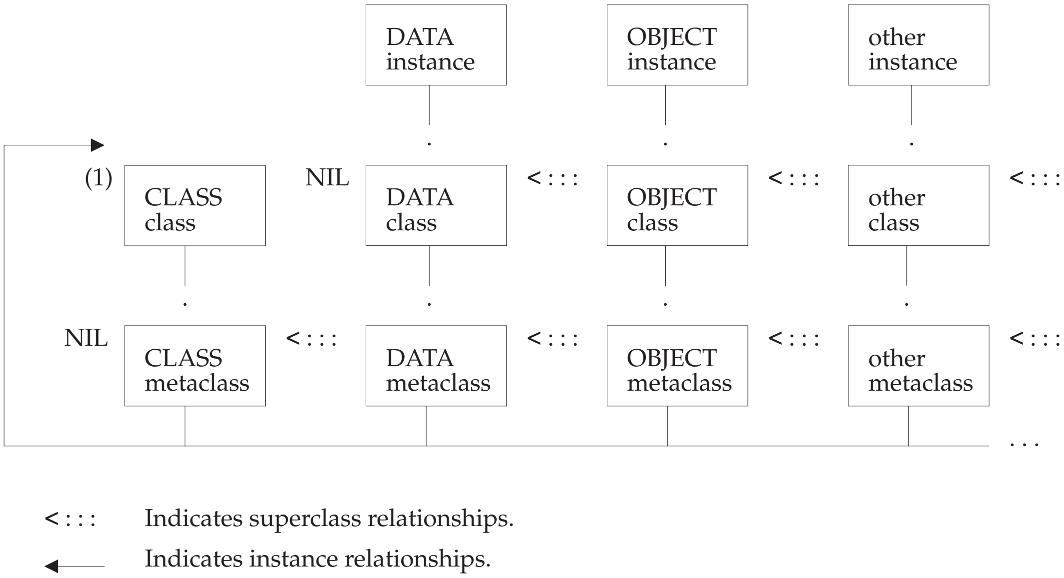


Figure 3-14 Superclass and Super-Metaclass Chains

Figure Notes

- Classes and metaclasses are combined as a single CLASS object in DDM.
- NIL is the end of a superclass chain.
- (1) 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 152 shows how a class instance would be mapped into memory as a collection of objects.

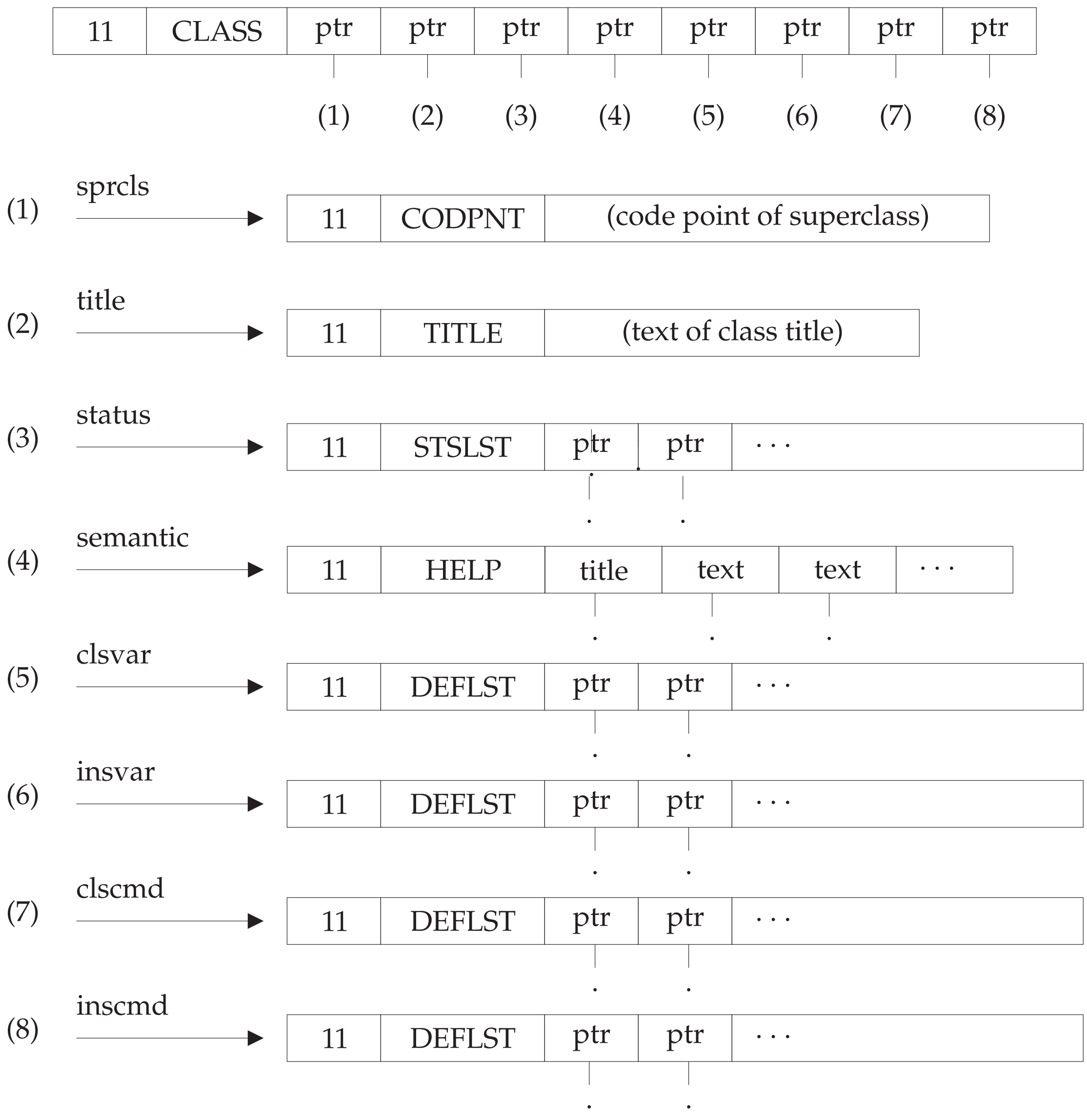


Figure 3-15 Class Structure Overview

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	BINDR - Binary Number Field
	LENGTH	16
	NOTE	The length (in bytes) of an object includes

		its own length and the length of all subsequent variables in the object.
class	INSTANCE_OF NOTE	CODPNTDR - Code Point Data Representation Specifies the code point of the class of the object.
sprcls	INSTANCE_OF NOTE	CODPNT - Code Point Specifies the code point of the superclass of the class being defined. The new class inherits variables and commands from its superclass. See description of the terms SPRCLS, INHERITANCE, and INHERITED.
title	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.
status	INSTANCE_OF TITLE NOTE	STSLST - Term Status Collection term status Specifies the status of the class.
semantic	INSTANCE_OF TITLE NOTE	CHRSTRDR - Character String description The semantic text of a class describes the class and provides information common to all instances of the class.
clsvar	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.
insvar	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.

clscmd	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 code point of its class.
inscmd	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 code point of its class.
insvar	See variable **clsvar** in this term.	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
class	*ABNUOWRM* on page 39
	ACCRDB on page 42
	ACCRDBRM on page 48
	ACCSEC on page 51
	ACCSECRD on page 55
	AGENT on page 56
	AGNPRMRM on page 61
	ARRAY on page 96
	ASSOCIATION on page 98
	ATTLST on page 99
	BGNBND on page 101
	BGNBNDRM on page 109
	BIN on page 110

Variable	Reference
	BINDR on page 112
	BITDR on page 115
	BITSTRDR on page 117
	BNDCHKEXS on page 119
	BNDCRTCTL on page 121
	BNDEXPOPT on page 123
	BNDOPT on page 127
	BNDOPTNM on page 128
	BNDOPTVL on page 129
	BNDSQLSTT on page 130
	BNDSTTASM on page 135
	BOOLEAN on page 136
	BYTDR on page 137
	CCSIDDBC on page 138
	CCSIDMBC on page 139
	CCSIDMGR on page 140
	CCSIDSBC on page 144
	CHRDR on page 145
	CHRSTRDR on page 146
	CLSQRY on page 163
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	CMDNSPRM on page 173
	CMDTRG on page 175
	CMDVLTRM on page 176
	CMMRQSRM on page 177

Variable	Reference
	CMMTYP on page 178
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	CNSVAL on page 216
	CNTQRY on page 217
	CODPNT on page 225
	CODPNTDR on page 228
	COLLECTION on page 231
	COMMAND on page 233
	CONSTANT on page 238
	CRRTKN on page 240
	DATA on page 245
	DCTIND on page 253
	DCTINDEN on page 254
	DECPRC on page 261
	DEFINITION on page 262
	DEFLST on page 263
	DEPERRCD on page 265
	DFTRDBCOL on page 268
	DFTVAL on page 269
	DGRIOPRL on page 270
	DICTIONARY on page 272
	DRPPKG on page 274
	DSCERRCD on page 277
	DSCINVRM on page 279
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DSS on page 289
	DSSFMT on page 301
	DTAMCHRM on page 303
	ELMCLS on page 306

Variable	Reference
	ENDBND on page 307
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	ENUCLS on page 316
	ENULEN on page 317
	ENUVAL on page 318
	EXCSAT on page 323
	EXCSATRD on page 329
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	EXTNAM on page 348
	FDODSC on page 354
	FDODSCOFF on page 356
	FDODTA on page 357
	FDODTAOFF on page 359
	FDOOBJ on page 360
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362
	FIELD on page 363
	HELP on page 371
	HEXDR on page 374
	HEXSTRDR on page 376
	IGNORABLE on page 378
	INHERITED on page 385
	INSTANCE_OF on page 387
	IPADDR on page 388
	LENGTH on page 398
	LOGNAME on page 410
	LOGTSTMP on page 411
	MANAGER on page 417
	MAXBLKEXT on page 420
	MAXLEN on page 422
	MAXRSLCNT on page 423
	MAXSCTNBR on page 424
	MAXVAL on page 425

Variable	Reference
	MGRDEPRM on page 426
	MGRLVL on page 427
	MGRLVLLS on page 429
	MGRLVLN on page 430
	MGRLVLRM on page 432
	MGRNAM on page 434
	MINLEN on page 439
	MINLVL on page 440
	MINVAL on page 441
	MTLEXC on page 442
	MTLINC on page 443
	NAMDR on page 444
	NAME on page 445
	NAMSYMDR on page 447
	NBRROW on page 449
	NEWPASSWORD on page 450
	NIL on page 451
	NOTE on page 452
	NUMBER on page 453
	OBJDSS on page 455
	OBJECT on page 459
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRY on page 475
	OPNQRYRM on page 483
	OPTIONAL on page 485
	ORDCOL on page 487
	OUTEXP on page 489
	PASSWORD on page 491
	PKGATHOPT on page 493
	PKGATHRUL on page 494
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGCNSTKN on page 500
	PKGDFTCC on page 501

Variable	Reference
	PKGDFTCST on page 503
	PKGID on page 504
	PKGISOLVL on page 505
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGOWNID on page 512
	PKGRPLOPT on page 516
	PKGRPLVRS on page 518
	PKGSN on page 519
	PKGSNLST on page 520
	PRCCNVCD on page 521
	PRCCNVRM on page 523
	PRCNAM on page 525
	PRDDTA on page 528
	PRDID on page 529
	PRMNSPRM on page 531
	PRPSQLSTT on page 533
	QLFATT on page 554
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	QRYDSC on page 560
	QRYDTA on page 561
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	QRYPRCTYP on page 566
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDB on page 571
	RDBACCCL on page 577
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBALWUPD on page 580
	RDBATHRM on page 581

Variable	Reference
	RDBCMM on page 582
	RDBCMTOK on page 585
	RDBCOLID on page 586
	RDBNACRM on page 587
	RDBNAM on page 589
	RDBNFNRM on page 592
	RDBRLLBCK on page 598
	RDBRLSOPT on page 603
	RDBUPDRM on page 604
	REBIND on page 606
	REPEATABLE on page 612
	REQUIRED on page 615
	RESERVED on page 617
	RLSCONV on page 619
	RPYDSS on page 620
	RPYMSG on page 624
	RQSCRR on page 626
	RQSDSS on page 629
	RSCLMTRM on page 634
	RSCNAM on page 637
	RSCTYP on page 638
	RSLSETFLG on page 639
	RSLSETRM on page 642
	RSNCOD on page 643
	RSYNCTYP on page 647
	RTNSQLDA on page 648
	SCALAR on page 649
	SECCHK on page 652
	SECCHKCD on page 656
	SECCHKRM on page 659
	SECMEC on page 661
	SECMGR on page 663
	SECMGRNM on page 666
	SECTKN on page 669
	SERVER on page 670

Variable	Reference
	SNAADDR on page 674
	SPCVAL on page 676
	SPRCLS on page 678
	SPVNAM on page 679
	SQLAM on page 694
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLCSRHLD on page 706
	SQLDARD on page 707
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLERRRM on page 710
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714
	SQLSTTNBR on page 715
	SQLSTTVRB on page 716
	SRVCLSNM on page 717
	SRVDGN on page 720
	SRVLCNT on page 722
	SRVLSRV on page 723
	SRVLST on page 725
	SRVNAM on page 727
	SRVPRTY on page 730
	SRVRLSLV on page 731
	STRING on page 735
	STTDATFMT on page 741
	STTDECDEL on page 742
	STTSTRDEL on page 744
	STTTIMFMT on page 746
	SUPERVISOR on page 753
	SVCERRNO on page 755
	SVRCOD on page 756
	SYNCCRD on page 759
	SYNCCTL on page 760

Variable	Reference
	SYNCLOG on page 764
	SYNCPTMGR on page 782
	SYNCRRD on page 826
	SYNCRSY on page 828
	SYNCTYPE on page 830
	SYNERRCD on page 831
	SYNTAXRM on page 835
	TCPHOST on page 846
	TEXT on page 864
	TITLE on page 865
	TRGDFTRT on page 867
	TRGNSPRM on page 868
	TYPDEF on page 872
	TYPDEFNAM on page 873
	TYPDEFOVR on page 876
	TYPFMLNM on page 880
	UOWDSP on page 882
	UOWID on page 883
	UOWSTATE on page 885
	USRID on page 888
	VALNSPRM on page 898
	VRSNAM on page 900
clsvar	*MANAGER* on page 417
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	COMMAND on page 233
	DICTIONARY on page 272
	DSS on page 289
	INHERITANCE on page 380
	INHERITED on page 385
	MANAGER on page 417
	MGROVR on page 436
	OBJECT on page 459
	OBJOVR on page 464
	SUBSETS on page 747

NAME

CLSQRY — Close Query

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'2005'
Length	*
Class	CLASS
Sprcls	COMMAND - Command

Close Query (CLSQRY) Command closes a query that an OPNQRY command opened.

Source System Processing

The source system determines the location of the RDB:

- Local: Call the local RDB server.
- Remote: Send the CLSQRY 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 *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 *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package. The *pkgnamcsn* has the same value as the *pkgnamcsn* parameter specified on the OPNQRY command which opened the query.

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 165 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 165). 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 CLSQRY Summary Decision Table

	Cases								
Conditions	**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 CLSQRY 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
PKGBPARM is returned		Y			Y				
Reply message is returned, see mdrpy			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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2005'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name

pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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	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'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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	ENDQRYRM on page 312
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYNOPRM on page 562
	SQL on page 680
	SQLAM on page 694

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'121C'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274

Variable	Reference
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63

CMDCHKRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1254'	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL ENUVAL 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 32 - ACCDMG - Access Damage Severity Code 64 - PRMDMG - Permanent Damage Severity Code 128 - SESDMG - Session Damage Severity Code
rdbnam	INSTANCE_OF MINLVL OPTIONAL NOTE	RDBNAM - Relational Database Name 3 Commands that operate on files do not return this parameter.
reccnt	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

srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
semantic	*DSS* on page 289
	EXCSQLSTT on page 336

CMDCMPRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'124B'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	TCPSRCCD on page 853

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 the description of the term SUBSETS.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1250'	
svrcod	INSTANCE_OF REQUIRED ENUVAL MINLVL NOTE ENUVAL	SVRCOD - Severity Code 4 - WARNING - Warning Severity Code 2 Can be used if wild-card characters were specified. 8 - ERROR - Error Severity Code
codpnt	INSTANCE_OF REQUIRED NOTE	CODPNT - Code Point Specifies the code point of the command not supported.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
semantic	*ACCSEC* on page 51
	AGNCMDPR on page 60
	OPTIONAL on page 485
	SUBSETS on page 747

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 code point identifier and the class of the target object.

Every command must have one and only one parameter with the CMDTRG attribute.

See the term AGNCMDPR for a description of command routing.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0041'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*AGNCMDPR* on page 60
	COMMAND on page 233
	SUBSETS on page 747

CMDVLTRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'221D'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*RDBCMM* on page 582
	RDBRLLBCK on page 598
semantic	*RDBCMM* on page 582
	RDBRLLBCK on page 598

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2225'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
cmmtyp	INSTANCE_OF REQUIRED	CMMTYP - Commitment Request Type
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336
semantic	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336

CMMTYP

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'2143'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	HEXSTRDR - Hexadecimal String 2 X'01' This value means commit was requested. X'02' This value means rollback was requested.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*CMMRQSRM* on page 177
semantic	*CMMRQSRM* on page 177

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:[5]

- 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.

5. 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 The 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 the description of the term APPCMNI.

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 is in the descriptions of the terms OBJDSS, RPYDSS, and RQSDSS.

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 the description of the term APPSRCCD. For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see the description of the term APPSRCCR.

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 the description of the term APPCMNT.

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 the description of the term APPSRCER.

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 the description of the term APPTRGER.

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 the description of the term APPCMNFL.

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 the description of the APPCMNI command).

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 (*APPCMNI* on page 68)

APPCMNT
: LU 6.2 Communications Termination (*APPCMNT* on page 72)

APPSRCCD
: LU 6.2 Source Command with Data (*APPSRCCD* on page 75)

APPSRCCR
: LU 6.2 Source Command Returning Data (*APPSRCCR* on page 82)

APPSRCER
: LU 6.2 Source Detected Error (*APPSRCER* on page 87)

APPTRGER
: LU 6.2 Target Detected Error (*APPTRGER* on page 91)

APPCMNFL
: LU 6.2 Communications Failure (*APPCMNFL* on page 64)

Transaction Program Names

- 07F0F0F1[6]
- All transaction program names beginning with 07F6
- A non-SNA-registered transaction program name the target system assigns to invoke the CMNAPPC communications manager

---

6. 07F0F0F1 was the only valid transaction program name (TPN) in DDM prior to DDM Level 3.

clsvar	**NIL**	
insvar	**NIL**	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	3[8]	
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
mgrdepls	*SYNCPTMGR* on page 782
semantic	*ACCRDB* on page 42
	AGENT on page 56
	APPCMNI on page 68
	CMNMGR on page 191
	CMNOVR on page 196
	CMNSYNCPT on page 197
	INHERITANCE on page 380
	RPYDSS on page 620
	RQSDSS on page 629
	SQLAM on page 694
	SUBSETS on page 747
	SYNCPTMGR on page 782

8. 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 190 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.

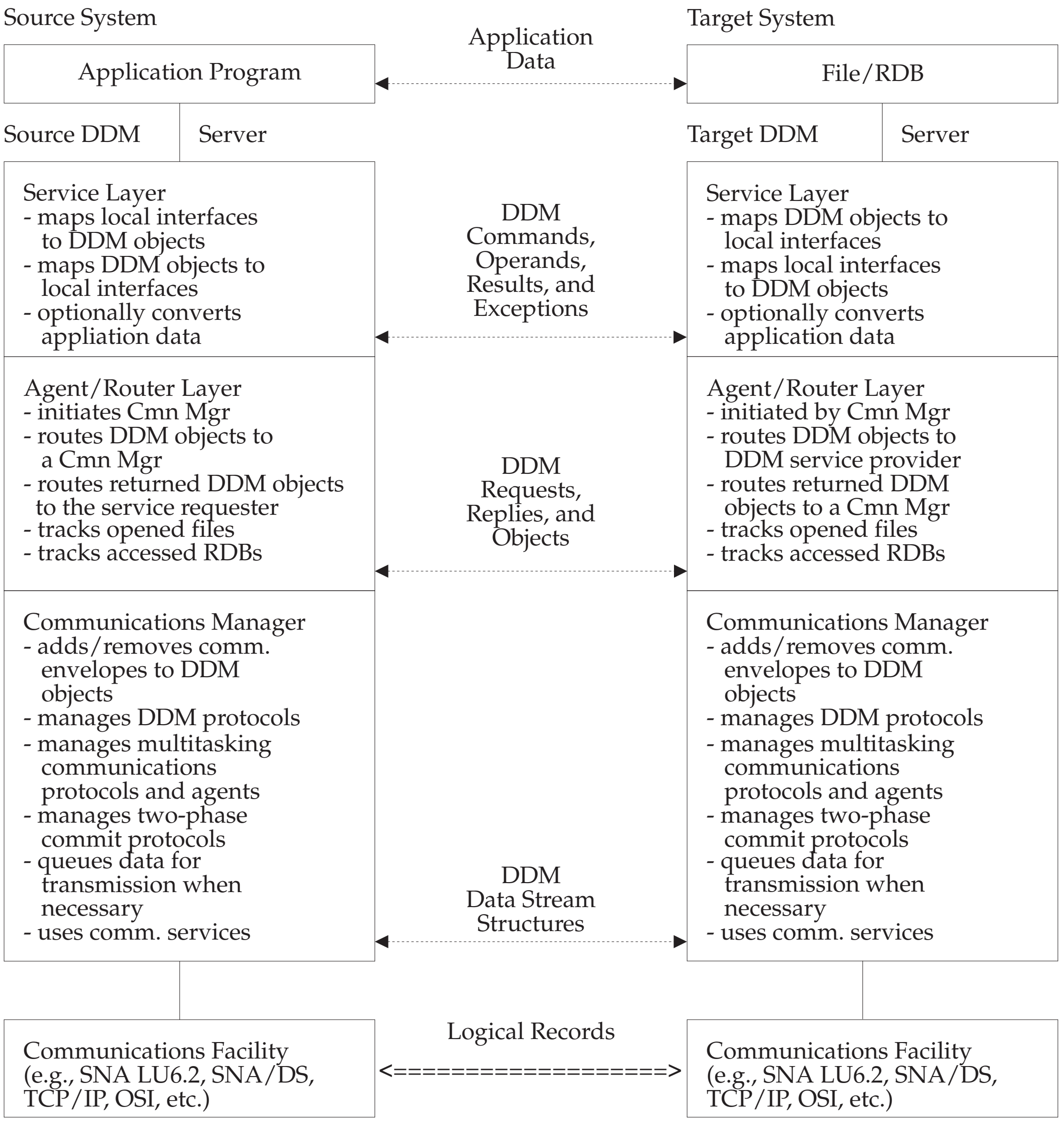


Figure 3-16 DDM as a Layered Communications Architecture

SEE ALSO

Variable	Reference
	CMNOVR on page 196
	CONCEPTS on page 237

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 DDM architecture. Its position in the server data paths is shown in Figure 3-17.

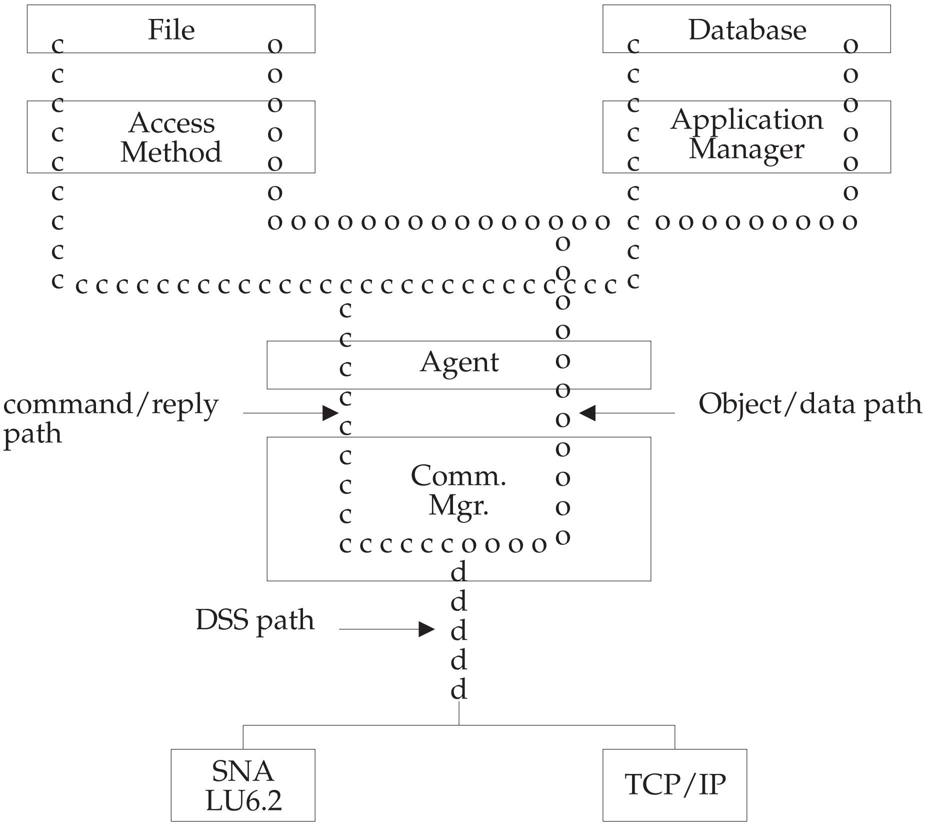


Figure 3-17 Server Data Paths for Access Method Services

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 the description of the term CMNAPPC for using SNA LU 6.2 for the DDM conversational protocol. See the description of the term CMNSYNCPT to find out how to use SNA LU 6.2 and sync point for the DDM conversational protocol. See the description of the term CMNTCPIP 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.

   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-overrun 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.[9]

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:

   1. SNA LU 6.2 Conversational Protocol (see description of the term CMNAPPC)
   2. SNA LU 6.2 Sync Point Conversational Protocol (see description of the term CMNSYNCPT)
   3. TCP/IP Conversational Protocol (see description of the term CMNTCPIP)

   Other DDM communications protocols will be defined as needed.

clsvar	**NIL**
insvar	**NIL**
clscmd	**NIL**
inscmd	**NIL**
mgrlvln	1
mgrdepls	**NIL**

9. The multitasking communications manager is introduced in DDM Level 4.

vldattls		VALID ATTRIBUTES
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
semantic	*CMNOVR* on page 196
	EXTENSIONS on page 346
	INHERITANCE on page 380
	MANAGER on page 417
	RDBOVR on page 593
sprcls	*CMNAPPC* on page 179
	CMNSYNCPT on page 197
	CMNTCPIP on page 209

CMNOVR

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 (*CMNLYR* on page 189)

CMNMGR

Communications Manager (*CMNMGR* on page 191)

CMNAPPC

LU 6.2 Conversational Communications Manager (*CMNAPPC* on page 179)

CMNSYNCPT

SNA LU 6.2 Sync Point Conversational Communications Manager (*CMNSYNCPT* on page 197)

CMNTCPIP

TCP/IP Communication Manager (*CMNTCPIP* on page 209)

SEE ALSO

Variable	Reference
semantic	*MGROVR* on page 436

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). More information on SNA LU 6.2 is in 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 199 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 The 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 the description of the term SYNCMNI.

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 is in the terms OBJDSS, RPYDSS, and RQSDSS.

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 the description of the term APPSRCCD. For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see the description of the term APPSRCCR.

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 the description of the term SYNCMNT.

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 the description of the term APPSRCER.

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 202.
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 202.

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 the description of the term APPTRGER.

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 the description of the term SYNCMNFL.

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 the term RESYNOVR 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 (*SYNCMNI* on page 774)

SYNCMNT
LU 6.2 Sync Point Communications Termination (*SYNCMNT* on page 778)

APPSRCCD
LU 6.2 Source Command with Data (*APPSRCCD* on page 75)

APPSRCCR
LU 6.2 Source Command Returning Data (*APPSRCCR* on page 82)

APPSRCER
LU 6.2 Source Detected Error (*APPSRCER* on page 87)

APPTRGER
LU 6.2 Target Detected Error (*APPTRGER* on page 91)

SYNCMNFL
LU 6.2 Sync Point Communications Failure (*SYNCMNFL* on page 771)

SYNCMNCM
LU 6.2 Sync Point Communications Two-Phase Commit (*SYNCMNCM* on page 768)

SYNCMNBK
LU 6.2 Sync Point Communications Backout (*SYNCMNBK* on page 766)

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

DDM Levels	1	2	3	4	5
CMNSYNCPT				4	4
Manager Dependencies					
SYNCPTMGR				4	4

clsvar	**NIL**	
insvar	**NIL**	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	4	
mgrdepls		**MANAGER DEPENDENCY LIST**
X'14C0'	INSTANCE_OF MGRLVLN NOTE	SYNCPTMGR - Sync Point Manager 4 The CMNSYNCPT must interface with the sync point manager for coordinated sync point services.
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
mgrdepls	*SYNCPTMGR* on page 782
semantic	*ACCRDB* on page 42
	AGENT on page 56
	CMNMGR on page 191
	CMNOVR on page 196
	INHERITANCE on page 380
	RDBCMM on page 582
	RDBOVR on page 593
	RDBRLLBCK on page 598
	RPYDSS on page 620
	RQSDSS on page 629
	SQLAM on page 694
	SUBSETS on page 747
	SYNCMNI on page 774
	SYNCPTMGR on page 782

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 extendable 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 the term TCPIPOVR (*TCPIPOVR* on page 847) 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 The 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 the description of TCPCMNI. 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 the description of TCPSRCCD. For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see the description of TCPSRCCR.

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 the description of TCPCMNT.

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 the description of TCPSRCER.

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 the description of TCPTRGER.

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 the description of TCPCMNFL.

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-BYTE	Access byte stream files using the DDM source and target servers with the byte stream access methods.

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 (*TCPCMNI* on page 840)

TCPCMNT
TCP/IP Communications Termination (*TCPCMNT* on page 844)

TCPSRCCD
TCP/IP Source Command with Data (*TCPSRCCD* on page 853)

TCPSRCCR
TCP/IP Source Command Returning Data (*TCPSRCCR* on page 856)

TCPSRCER
TCP/IP Source Detected Error (*TCPSRCER* on page 859)

TCPTRGER
TCP/IP Target Detected Error (*TCPTRGER* on page 861)

TCPCMNFL
TCP/IP Communications Failure (*TCPCMNFL* on page 838)

---

clsvar **NIL**

insvar	**NIL**	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	5	
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
	SYNCLOG on page 764
mgrdepls	*SYNCPTMGR* on page 782
semantic	*AGENT* on page 56
	CMNMGR on page 191
	RPYDSS on page 620
	RQSDSS on page 629
	SUBSETS on page 747
	SYNCPTMGR on page 782
	TCPHOST on page 846
	TCPIPOVR on page 847

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'1070'	
value	INSTANCE_OF LENGTH	BYTSTRDR - Byte String 4
clscmd	**NIL**	
inscmd	**NIL**	

CNSVAL

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘000B’	
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’.
clscmd	**NIL**	
inscmd	**NIL**	

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 (for detailed information, see the description of the terms FIXROWPRC and LMTBLKPRC).

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.

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 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 *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 *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.

The *qryrelscr* parameter controls whether to use the relative scrolling action for scrollable cursors. If absolute scrolling is desired, specify FALSE for the value of this parameter.

The *qryrownbr* parameter specifies the row (absolute) or how many rows from the current row (relative) to begin fetching. For absolute positioning, the first row is +1, the last is −1. For relative positioning, +1 is the next row, −1 is the previous row, and so on.

The *qryrfrtbl* parameter indicates to the target SQLAM to recompute the answer set table prior to returning the row(s) of data requested.

The *nbrrow* parameter specifies the number of data rows being requested (FETCHed).

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 CNTQRY command is valid any time that the target SQLAM suspends a query or result set.

Normal completion of this command returns one or more QRYDTA reply data objects. The QRYDSC objects, which were returned when the query was opened or when the stored procedure completed, describe the QRYDTA reply data objects (see the terms OPNQRY and EXCSQLSTT).

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 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.

Table 3-10 on page 220 is a decision table that summarizes the CNTQRY command 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 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 object 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 object are returned if enough space is available in the last query block; otherwise, they are saved and not returned.

An ENDQRYRM and an SQLCARD object are returned after any QRYDTA reply data objects, and the query is terminated if:

- The last of the answer set data is returned, or some RDB detected error condition has occurred.
- LMTBLKPRC query protocols and FIXROWPRC query protocols with a blocked fetch are used.
- Space is available in the current query block for an ENDQRYRM and SQLCARD object.

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 all of the answer set data is returned for the current CNTQRY command, then an ENDQRYRM and SQLCARD object are returned with the answer set data, and the query or result set is terminated.

If all of the answer set data is returned on the previous command, then an ENDQRYRM and SQLCARD object are returned without the answer set data, and the query or result set is terminated.

If the bind process is active, then the command is rejected with the PKGBPARM, 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.

Summary Decision Table

The CNTQRY functions are summarized in the decision table shown in Table 3-10 on page 220. The decision table only contains information already presented in the previous text. The decision table has been included as a development aid.

Table 3-10 CNTQRY Summary Decision Table (Part 1)

Conditions	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Query is in the not-opened state	X	X	X												
Query is in the suspended state				X	X	X	X	X	X	X	X	X	X	X	X
Package binding process is active					X										
Last row of data was returned by the preceding OPNQRY or CNTQRY command						X	X	X							
Last row of data is returned by this CNTQRY command[10]									X	X	X				
Reply message and SQLCARD were saved by the preceding OPNQRY or CNTQRY command							X					X			
Reply message and SQLCARD are returned in the last query block									X				X		
Reply message and SQLCARD are not returned in the last query block										X				X	
Current unit of work is rolled back, possibly due to a deadlock situation		X						X			X				
Error detected preventing the CNTQRY command from being initiated by the SQLAM			X												X
RDB had an error preventing the query from resuming													X	X	

Table 3-11 CNTQRY Summary Decision Table (Part 2)

	Cases														
Actions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**N**	**M**	**O**
Query is in the suspended state				Y	Y					Y	Y			Y	Y
One or more QRYDTAs are returned				Y					Y	Y	Y		Y	Y	
SQLCARD is returned		Y				Y		Y	Y				Y		
Saved reply message and SQLCARD are returned from the preceding OPNQRY or CNTQRY command							Y					Y			
Reply message and SQLCARD are saved										Y	Y			Y	
QRYNOPRM is returned	Y														
Query is terminated, placed in the not-opened state	Y	Y	Y			Y	Y	Y	Y			Y	Y		
PKGBPARM is returned					Y										
ENDQRYRM is returned						Y			Y				Y		
Reply message is returned, see mdrpy			Y												Y
ABNUOWRM is returned		Y						Y							

Note: A column specifies a case, consisting of a set of conditions and a set of actions resulting from receiving the CNTQRY command. 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*.

Source System Reply Processing

Return any reply messages and data objects to the requester.

---

10. This row may not be complete in the case of a RLLBCK condition.

10. For FIXROWPRC with a single row fetch, the reply message and SQLCARD are never placed in the query block containing the single row of data.

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'2006'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
qryblksz	INSTANCE_OF REQUIRED	QRYBLKSZ - Query Block Size
qryrelscr	INSTANCE_OF OPTIONAL	QRYRELSCR - Query Relative Scrolling Action
qryrownbr	INSTANCE_OF OPTIONAL	QRYROWNBR - Query Row Number
qryrfrtbl	INSTANCE_OF OPTIONAL	QRYRFRTBL - Query Refresh Answer Set Table
nbrrow	INSTANCE_OF OPTIONAL	NBRROW - Number of Fetch or Insert Rows
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks 0
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4

	DFTVAL	′′
	NOTE	The default means the value received on ACCRDB command is used.
	NOTE	If sent, the value does not apply to the data in the QRYDTA objects.
X′0035′	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	OPTIONAL	
	DFTVAL	′′
	NOTE	The default means the value received on ACCRDB command is used.
	NOTE	If sent, the value does not apply to the data in the QRYDTA objects.
X′2408′	INSTANCE_OF	SQLCARD - SQL Communications Area Reply Data
	OPTIONAL	
	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	In the nonterminating error condition, the QRYDTA object contains only an SQLCA that reports the error.
cmdrpy		**COMMAND REPLIES**
X′220D′	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
	NOTE	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′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

Variable	Reference
inscmd	*SQLAM* on page 694
rpydta	*OPNQRY* on page 475
semantic	*APPSRCCR* on page 82
	BGNBND on page 101
	CLSQRY on page 163
	ENDQRYRM on page 312
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	MAXBLKEXT on page 420
	OPNQRY on page 475
	QRYBLK on page 555
	QRYBLKSZ on page 559
	QRYNOPRM on page 562
	QRYRELSCR on page 567
	QRYROWNBR on page 569
	SQL on page 680
	SQLAM on page 694

NAME

CODPNT — Code Point

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Codepoint X'000C'

Length *

Class CLASS

Sprcls HELP - Help Text

The Code Point (CODPNT) Data specifies a scalar value that is an architected code point.

See the description of the term CODPNTDR for information on the representation and use of code point data.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'000C'	
title	INSTANCE_OF	TITLE - Title
status	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
semantic	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 description
codpnt	INSTANCE_OF	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
class	*BNDCHKONL* on page 120
	BNDERRALW on page 122
	BNDEXSOPT on page 124
	BNDEXSRQR on page 125
	BNDNERALW on page 126
	CSTBITS on page 241
	CSTMBCS on page 242
	CSTSBCS on page 243

Variable	Reference
	CSTSYSDFT on page 244
	DECDELCMA on page 259
	DECDELPRD on page 260
	DFTPKG on page 267
	EURDATFMT on page 321
	EURTIMFMT on page 322
	EXPALL on page 344
	EXPNON on page 345
	FIXROWPRC on page 365
	FRCFIXROW on page 370
	ISODATFMT on page 389
	ISOLVLALL on page 390
	ISOLVLCHG on page 391
	ISOLVLCS on page 392
	ISOLVLNC on page 393
	ISOLVLRR on page 394
	ISOTIMFMT on page 395
	JISDATFMT on page 396
	JISTIMFMT on page 397
	LMTBLKPRC on page 400
	PKGATHKP on page 492
	PKGATHRVK on page 496
	PKGRPLALW on page 514
	PKGRPLNA on page 515
	RDBRLSCMM on page 601
	RDBRLSCNV on page 602
	STGLMT on page 732
	STRDELAP on page 733
	STRDELDQ on page 734
	STTASMEUI on page 740
	STTSCCCLS on page 743
	USADATFMT on page 886
	USATIMFMT on page 887
clsvar	*CLASS* on page 148
insvar	*CMDNSPRM* on page 173

Variable	Reference
	DCTINDEN on page 254
	OBJNSPRM on page 462
	PRMNSPRM on page 531
	SYNTAXRM on page 835
	VALNSPRM on page 898
semantic	*AGNCMDPR* on page 60
	CLASS on page 148

NAME

CODPNTDR — Code Point Data Representation

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Codepoint X'0064'

Length *

Class CLASS

Sprcls FIELD - A Discrete Unit of Data

Code Point Data Representation (CODPNTDR) specifies the data representation of a dictionary code point. Code points are hexadecimal aliases for DDM architecture's named terms. Code points reduce the number of bytes required to identify the class of an object in memory and in data streams.

Figure 3-18 on page 229 illustrates how the code points of objects stored in memory or received from a data stream refer to the class descriptions stored in a dictionary for those objects.

Code points 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 code points to class descriptors is illustrated in Figure 3-18 on page 229.

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 the description of the term EXTENSIONS)
- 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 code points 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.

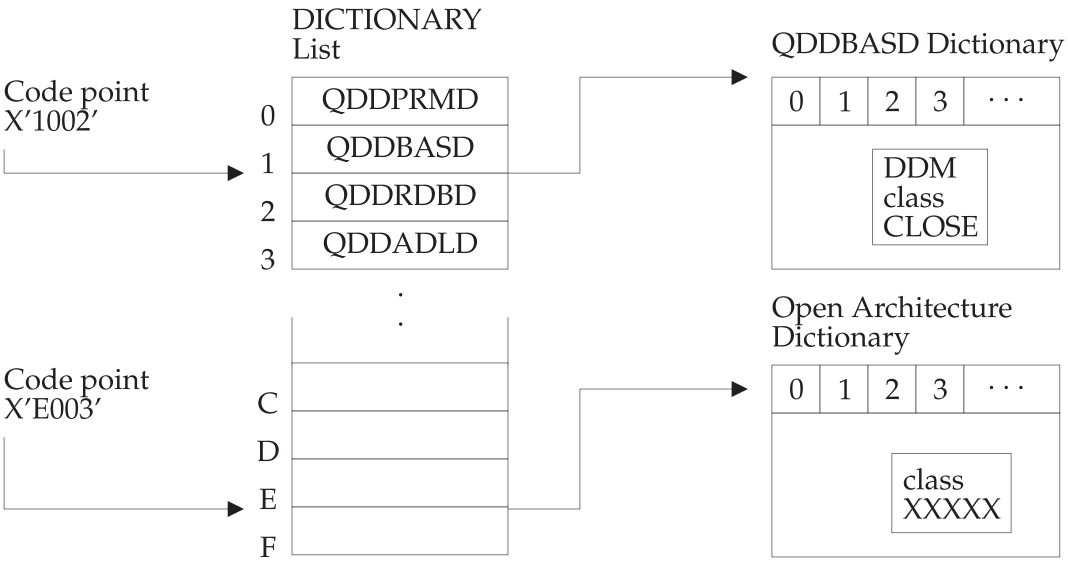


Figure 3-18 Code Point References to Dictionaries

Length Specification

The length of CODPNTDR fields is two bytes.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
index	INSTANCE_OF NOTE	HEXDR - Hexadecimal Number Specifies an index into the dictionary list of the agent.
identifier	INSTANCE_OF LENGTH NOTE	HEXSTRDR - Hexadecimal String 3 Specifies a unique identifier associated with a specific term in a dictionary.
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
insvar	*BNDCHKEXS* on page 119
	BNDCRTCTL on page 121
	BNDEXPOPT on page 123

Variable	Reference
	BNDSTTASM on page 135
	CODPNT on page 225
	ELMCLS on page 306
	ENUCLS on page 316
	INSTANCE_OF on page 387
	MGRLVL on page 427
	MTLEXC on page 442
	MTLINC on page 443
	OBJECT on page 459
	PKGATHOPT on page 493
	PKGDFTCST on page 503
	PKGISOLVL on page 505
	PKGRPLOPT on page 516
	QRYBLKCTL on page 557
	QRYPRCTYP on page 566
	RDBACCCL on page 577
	RDBRLSOPT on page 603
	RSCTYP on page 638
	SPRCLS on page 678
	STTDATFMT on page 741
	STTDECDEL on page 742
	STTSTRDEL on page 744
	STTTIMFMT on page 746
semantic	*CODPNT* on page 225
	EXTENSIONS on page 346

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.

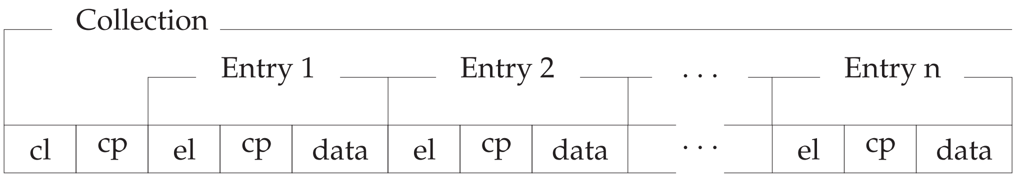


Figure 3-19 Collection-to-Data-Stream Mapping

Legend

cl Length of collection.

cp A code point.

el Entry length.

data Entry data.

All objects in DDM are modeled as either scalars (see description of the term SCALAR) or collections. Collections are modeled as a structure with the following fields:

- Length—the total length of the collection, including the length and code point fields (four bytes), and the length of all entries in the collection.

  See the term DSS 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 code point 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 *.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	4	
class	X'000D'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*DSS* on page 289
	INHERITANCE on page 380
	OBJECT on page 459
	OBJOVR on page 464
sprcls	*ACCSECRD* on page 55
	ARRAY on page 96
	ATTLST on page 99
	BNDOPT on page 127
	CLASS on page 148
	COMMAND on page 233
	EXCSATRD on page 329
	ORDCOL on page 487
	PKGDFTCC on page 501
	RPYMSG on page 624
	SRVLSRV on page 723
	SRVLST on page 725
	SYNCCRD on page 759
	SYNCRRD on page 826
	TYPDEFOVR on page 876

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 code points.
   - 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 the terms AGNCMDPR and CMDTRG 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))
```

clsvar		**CLASS VARIABLES**
cmddta	INSTANCE_OF TITLE NOTE	DEFLST - Definition List command data objects All commands can specify the data objects to be transmitted in OBJDSSs.
rpydta	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.
cmdrpy	INSTANCE_OF TITLE NOTE	DEFLST - Definition List command replies All commands can specify the reply messages that can be generated for them.
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'000E'	
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta	**NIL**	
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	DSS on page 289
	INHERITANCE on page 380
	RQSDSS on page 629
	SUBSETS on page 747
sprcls	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652

Variable	Reference
	SYNCCTL on page 760
	SYNCRSY on page 828

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 (*ABBREVIATIONS* on page 30)

CMNLYR
Communications Layers (*CMNLYR* on page 189)

DDMOBJ
DDM Objectives

DSS Data Stream Structures (*DSS* on page 289)

EXTENSIONS
Product Extensions to the DDM Architecture (*EXTENSIONS* on page 346)

INHERITANCE
Class Inheritance (*INHERITANCE* on page 380)

LVLCMP
Level Compatibility (*LVLCMP* on page 412)

OOPOVR
Overview of Object-Oriented Programming

PRCOVR
DDM Processing Overview (*PRCOVR* on page 526)

STRLYR Structural Layers of the DDM Architecture (*STRLYR* on page 737)

SUBSETS
Architecture Subsets (*SUBSETS* on page 747)

TASK Task

SEE ALSO

Variable	Reference
semantic	*DDM* on page 255

CONSTANT

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0050'	
title	INSTANCE_OF	TITLE - Title
status	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
semantic	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 description
value	**NIL**	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
class	*ACCDMG* on page 41
	DCESEC on page 247
	DDMID on page 258
	ERROR on page 319
	FALSE on page 349
	INFO on page 379
	OWNER on page 490

Variable	Reference
	PRMDMG on page 530
	REQUESTER on page 614
	SESDMG on page 672
	SEVERE on page 673
	TRUE on page 870
	USRIDNWPWD on page 889
	USRIDONL on page 890
	USRIDPWD on page 891
	WARNING on page 902

CRRTKN

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2135'	
value	INSTANCE_OF MINLEN MAXLEN REQUIRED	BYTSTRDR - Byte String 0 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
	ACCRDBRM on page 48
semantic	*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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	PKGDFTCST on page 503
semantic	*ENDBND* on page 307

CSTMBCS

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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	PKGDFTCST on page 503

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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	PKGDFTCST on page 503

CSTSYSDFT

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

Variable	Reference
insvar	*PKGDFTCST* on page 503

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.

clsvar	**NIL**
insvar	**NIL**
clscmd	**NIL**
inscmd	**NIL**

SEE ALSO

Variable	Reference
semantic	*APPCMNFL* on page 64
	APPCMNI on page 68
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CLASS on page 148
	CSTBITS on page 241
	CSTMBCS on page 242
	CSTSBCS on page 243
	INHERITANCE on page 380
	STRLYR on page 737
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771

Variable	Reference
	SYNCMNI on page 774
	SYNCMNT on page 778
sprcls	*DSS* on page 289
	FIELD on page 363
	HELP on page 371
	MGRLVL on page 427
	OBJECT on page 459
	RQSCRR on page 626

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:

OSFDCE
OSF DCE Security Mechanism (*OSFDCE* on page 488)

DCESECOVR
DCE Security Overview (*DCESECOVR* on page 248)

DCESECTKN
DCE Security Token (*DCESECTKN* on page 252)

value	1

SEE ALSO

Variable	Reference
cmddta	*SECCHK* on page 652
insvar	*SECMEC* on page 661
semantic	*DCESECOVR* on page 248
	SECCHK on page 652
	SECMEC on page 661

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 249 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)*

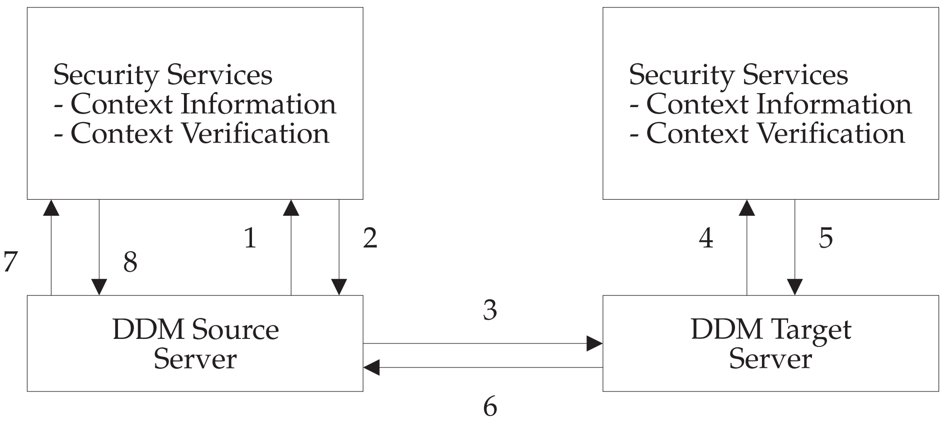


Figure 3-20 DCE-Based Security Flows Using GSS-API

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.

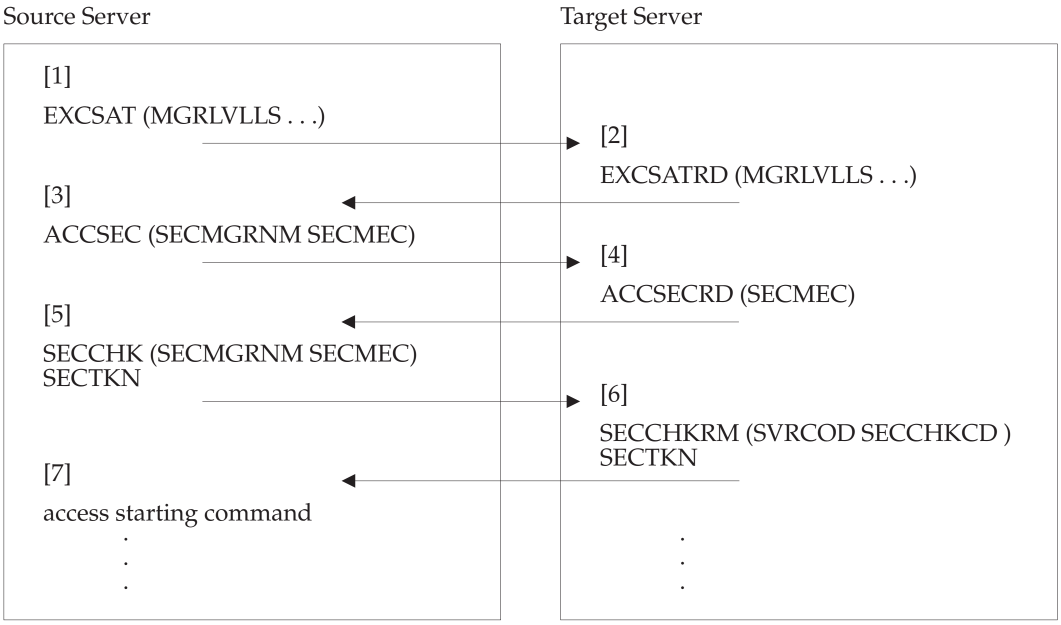


Figure 3-21 DDM DCE Security Flows

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 DDM term DCESECTKN.

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 249 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

Variable	Reference
semantic	*DCESEC* on page 247
	DCESECTKN on page 252
	SECCHK on page 652
	SECMGR on page 663
	SECOVR on page 667

DCESECTKN

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 the term *DCESECOVR* on page 248 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

Variable	Reference
semantic	*DCESEC* on page 247
	DCESECOVR on page 248

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 code points to the ordinal numbers of selected dictionaries. Not all dictionary objects have external names or code points.

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 code point. Local data managers can use other structures such as binary radix trees instead of this model to improve performance.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1450'	
entry	INSTANCE_OF REPEATABLE	DCTINDEN - Dictionary Index Entry
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DICTIONARY* on page 272
semantic	*DICTIONARY* on page 272

DCTINDEN

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 code point with the ordinal number where the object is stored in the dictionary.

insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1451'	
key	ENUCLS ENUCLS REQUIRED	NAME - Name CODPNT - Code Point
value	INSTANCE_OF LENGTH REQUIRED	BIN - Binary Integer Number 32
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DCTIND* on page 253
semantic	*DICTIONARY* on page 272

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:

CONCEPTS
Concepts of the DDM Architecture (*CONCEPTS* on page 237)

SRVOVR
Server Layer Overview (*SRVOVR* on page 728)

MGROVR
Manager Layer Overview (*MGROVR* on page 436)

OBJOVR
Object Layer Overview (*OBJOVR* on page 464)

DTAOVR
Data Layer Overview (*DTAOVR* on page 305)

SEE ALSO

None.

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

None.

DDMID

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

Variable	Reference
insvar	*OBJDSS* on page 455
	RPYDSS on page 620
	RQSDSS on page 629
semantic	*DSS* on page 289

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

Variable	Reference
insvar	*STTDECDEL* on page 742

DECDELPRD

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

Variable	Reference
insvar	*STTDECDEL* on page 742
semantic	*ENDBND* on page 307

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2106'	
value	INSTANCE_OF LENGTH ENUVAL DFTVAL NOTE	BINDR - Binary Number Field 16 15 '' The default value means that the target RDB selects the target product-specific decimal precision for arithmetic computations.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	ENDBND on page 307

DEFINITION

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0048'	
key	ELMCLS REQUIRED	NAME - Name
value	SPRCLS REQUIRED	ATTLST - Attribute List
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DEFLST* on page 263
semantic	*DEFLST* on page 263

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.

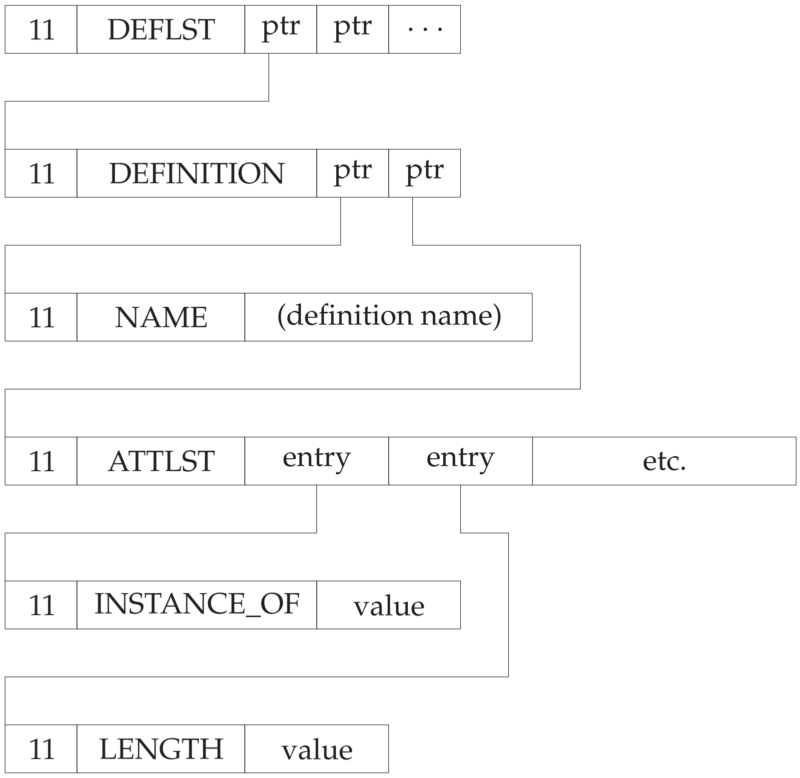


Figure 3-22 Structure of a Definition List

clsvar	**NIL**

insvar		CLASS INSTANCE VARIABLES
length	*	
class	X'0047'	
entry	INSTANCE_OF REPEATABLE REQUIRED	DEFINITION - Definition
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
	COMMAND on page 233
	FIELD on page 363
	MANAGER on page 417
semantic	*CLASS* on page 148
	INHERITED on page 385

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.

clsvar	**NIL**	
insvar		**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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*MGRDEPRM* on page 426
semantic	*MGRDEPRM* on page 426

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

Variable	Reference
insvar	*STTDECDEL* on page 742
	STTSTRDEL on page 744

DFTRDBCOL

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2128'	
name	INSTANCE_OF	NAMSYMDR - Name Symbol Data Representation
	MAXLEN	18
	DFTVAL	''
	NOTE	The default value means that the target RDB selects a default RDB collection ID value used for RDB object name completion.
	OPTIONAL	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	REBIND on page 606

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, code point, and value) need not be sent.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0011'	
value	SPRCLS NOTE SPCVAL NOTE REQUIRED	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'. '' The receiving server determines the default value.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*LVLCMP* on page 412

DGRIOPRL

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

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101

Variable	Reference
	REBIND on page 606

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 code point 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 code points.

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 code point of the object. For collection objects, the data following the length and class code point 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.

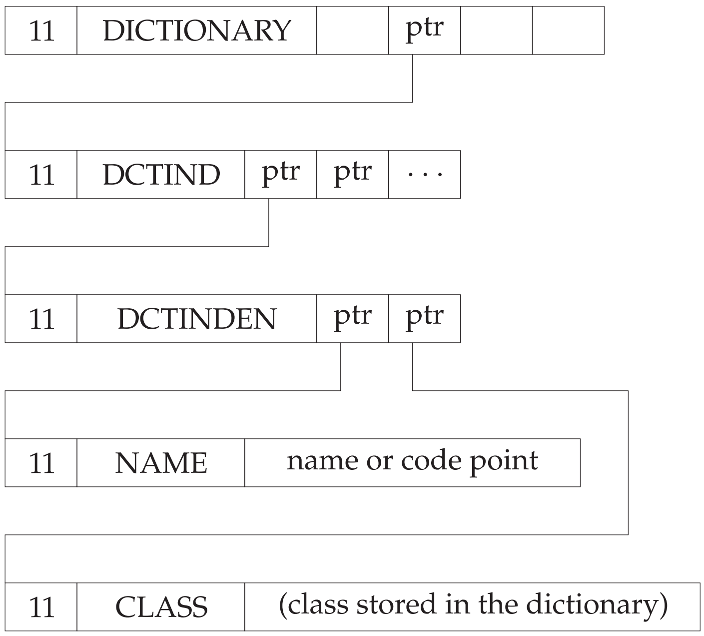


Figure 3-23 Dictionary Structure Overview

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1458'	
dctind	INSTANCE_OF	DCTIND - Dictionary Index
objlst	*	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	1	
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*MGRLVL* on page 427
mgrdepls	*AGENT* on page 56
semantic	*AGENT* on page 56
	CODPNTDR on page 228
	DCTIND on page 253
	EXTENSIONS on page 346
	INHERITANCE on page 380
	MANAGER on page 417
	MGROVR on page 436
	OBJOVR on page 464
	RDBOVR on page 593
	SUBSETS on page 747

DRPPKG

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 *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 *pkgnam* parameter specifies the fully qualified package name of the package to be dropped from the RDB.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2007'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnam	INSTANCE_OF REQUIRED	PKGNAM - RDB Package Name
vrsnam	INSTANCE_OF OPTIONAL	VRSNAM - Version Name
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 '' The default means the value received on ACCRDBRM is used.

	OPTIONAL	
X'0035'	INSTANCE_OF DFTVAL NOTE OPTIONAL	TYPDEFOVR - TYPDEF Overrides '' The default means the value received on ACCRDBRM is used.
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data

cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	SQL on page 680
	SQLAM on page 694

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.[11]

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 Distributed Relational Database Architecture (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.

11. 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 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.
value (continued>	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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DSCINVRM* on page 279
semantic	*DSCINVRM* on page 279

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'220A'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
dscerrcd	INSTANCE_OF REQUIRED	DSCERRCD - Description Error Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
fdodsc	INSTANCE_OF REQUIRED	FDODSC - FD:OCA Data Descriptor
fdodscoff	INSTANCE_OF REQUIRED	FDODSCOFF - FD:OCA Descriptor Offset
fdotrpoff	INSTANCE_OF REQUIRED	FDOTRPOFF - FD:OCA Triplet Offset
fdoprmoff	INSTANCE_OF REQUIRED	FDOPRMOFF - FD:OCA Triplet Parameter Offset

srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSQLSTT* on page 336
	OPNQRY on page 475
semantic	*EXCSQLSTT* on page 336
	OPNQRY on page 475

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 *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 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

Exception conditions that the RDB detects are reported in the SQLDARD or SQLCARD object.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘2012’	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
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
rpydta		**REPLY OBJECTS**
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'2411'	INSTANCE_OF REQUIRED MTLEXC	SQLDARD - SQLDA Reply Data X'2408' - SQLCARD - SQL Communications Area Reply Data
X'2408'	INSTANCE_OF REQUIRED MTLEXC NOTE	SQLCARD - SQL Communications Area Reply Data X'2411' - SQLDARD - SQLDA Reply Data Only returned if an ABNUOWRM is also returned.
cmdrpy		**COMMAND REPLIES**
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'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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	SQL on page 680
	SQLAM on page 694

NAME

DSCSQLSTT — Describe SQL Statement

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2008'

Length *

Class CLASS

Sprcls COMMAND - Command

Describe SQL Statement (DSCSQLSTT) Command returns the definitions of the select list 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.

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 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.

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.

Normal completion of this command results in the select list data variable definitions being returned in the SQLDARD reply data object. The SQLDARD FD:OCA descriptor describes the 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

Exception conditions that the RDB detects are reported in the SQLDARD or 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2008'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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'2411'	INSTANCE_OF REQUIRED MTLEXC	SQLDARD - SQLDA Reply Data X'2408' - SQLCARD - SQL Communications Area Reply Data
X'2408'	INSTANCE_OF REQUIRED MTLEXC NOTE	SQLCARD - SQL Communications Area Reply Data X'2411' - SQLDARD - SQLDA Reply Data Returned only if an ABNUOWRM is also returned.
cmdrpy		**COMMAND REPLIES**
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'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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	SQL on page 680
	SQLAM on page 694

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.

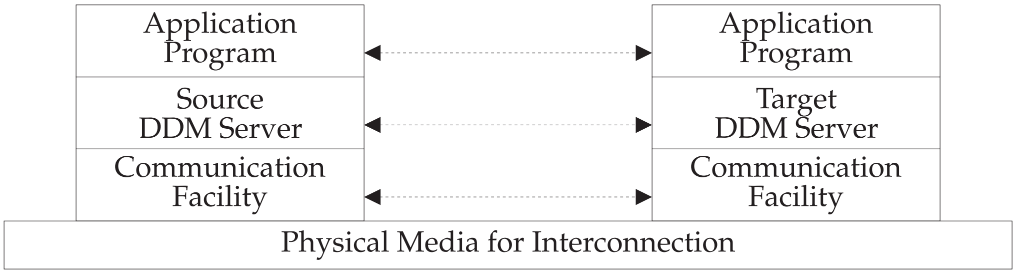


Figure 3-24 Layered Communications Reference Model

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 290:

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).

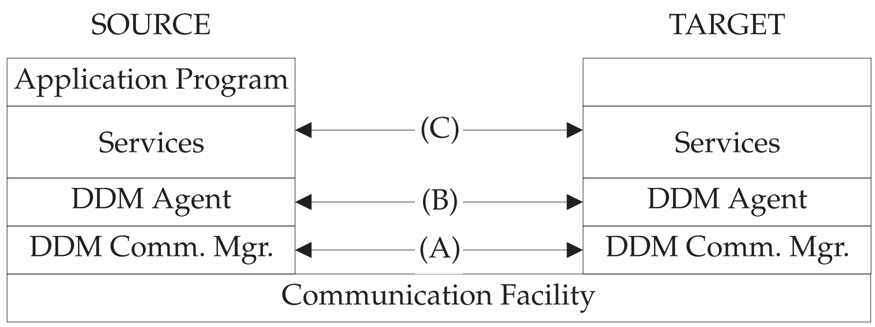


Figure 3-25 Layers (A, B, and C) in the DDM Architecture

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 the term DDMID). This is illustrated in Figure 3-26.

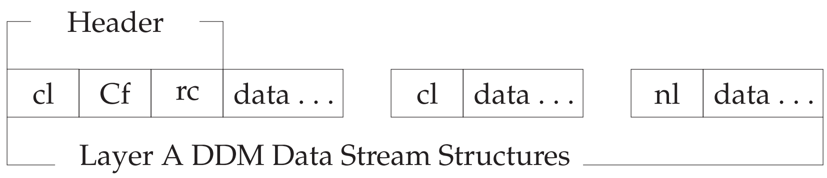


Figure 3-26 DDM Layer A Data Stream Structure

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 292 and Figure 3-28 on page 293for DSS chaining rules.
- Continuation or termination of chains when errors are detected

  See Figure 3-28 on page 293 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 295 and Figure 3-31 on page 296.

The *f* byte qualifier is used to specify (see the description of the term DSSFMT):

- Whether the structure is a communications structure, 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.
- 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 the description of the term RQSCRR for a discussion of correlation.

For the communications structure, the two-byte identifier is redefined into a subtype of the communications message. See the description of the term CTLFMT.

Four 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 the description of the term RQSDSS).
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 the description of the term RPYDSS).
3. Object structures are used for all objects (for instance, records) sent between DDM Agents (see the description of the term OBJDSS).
4. Communications structures are used between the communication managers to control the communications flows between the source and target communication managers. For instance, the communication manager uses the CMNDSS and the CMNMSG to control interleaving and multi-leaving protocols for multitasking.

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.

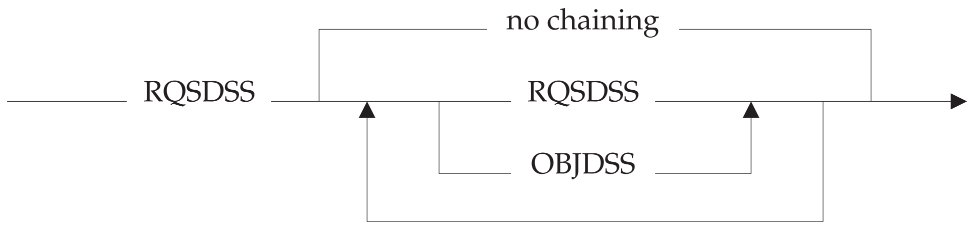


Figure 3-27 RQSDSS Chaining Rules

RQSDSS Chaining Rules

The following rules apply to Figure 3-27 on page 292:

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.

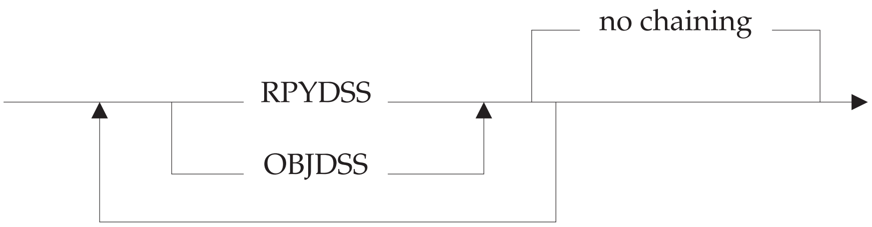


Figure 3-28 RPYDSS and OBJDSS Chaining Rules

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		RPYDSS Chain	
Status	**DSSs**	**DSSs**	**Severity**
done	RQSDSS 1	RPYDSS 1	0
done	RQSDSS 2	OBJDSS 2	
		RPYDSS 2 4	
in process	RQSDSS 3	RPYDSS 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 SEVER (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 on page 295. 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 (32763 bytes) long consist of:

- A two-byte length field
- A two-byte class field that identifies the class (type) of an object by its code point
- 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 on page 295. Note that the object can be segmented at Layer A to accommodate transmission buffer restrictions (or for any other reason).

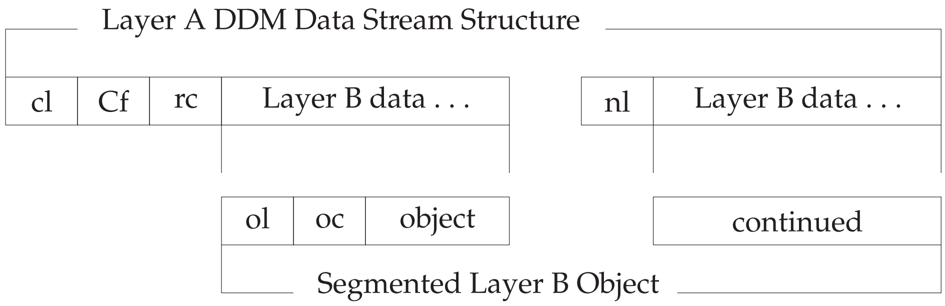


Figure 3-30 Mapping Small DDM Layer B Objects to Layer A DSSs

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 code point.

Most DDM objects are small, but for objects with data greater than 32 Kbytes less 5 bytes (32763 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 on page 296. Note that the object can be segmented at Layer A to accommodate transmission buffer restrictions (or for any other reason).

An alternative solution for handling large objects is book ends. See the description in the term BKENDOVR for an explanation of book ends.

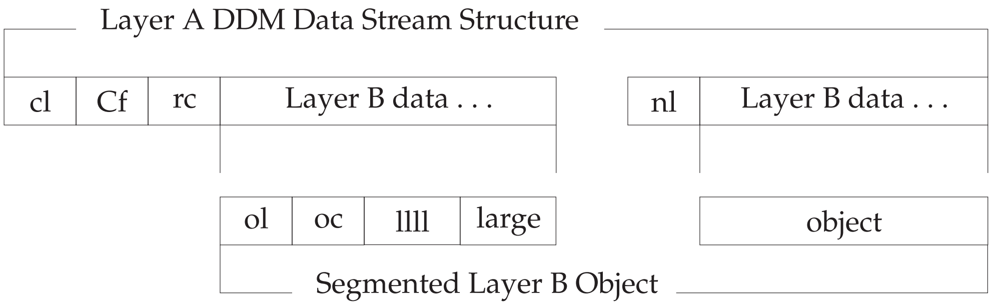


Figure 3-31 Mapping Large DDM Layer B Objects to Layer A DSSs

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 code point.

llll Extended total 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 on page 297.

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

Variable	Reference
	DSSFMT on page 301
sprcls	*OBJDSS* on page 455
	PRCCNVCD on page 521
	RPYDSS on page 620
	RQSDSS on page 629
	SYNERRCD on page 831
semantic	*ABNUOWRM* on page 39
	ACCRDB on page 42
	ACCRDBRM on page 48
	ACCSEC on page 51
	AGNPRMRM on page 61
	APPSRCCD on page 75
	APPSRCER on page 87
	BGNBND on page 101
	BGNBNDRM on page 109
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172

Variable	Reference
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	CNTQRY on page 217
	COLLECTION on page 231
	CONCEPTS on page 237
	DRPPKG on page 274
	DSCINVRM on page 279
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DSSFMT on page 301
	DTAMCHRM on page 303
	ENDBND on page 307
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	EXCSAT on page 323
	EXCSATRD on page 329
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FDOOBJ on page 360
	MANAGER on page 417
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJECT on page 459
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRY on page 475
	OPNQRYRM on page 483

Variable	Reference
	PKGBNARM on page 497
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	PRPSQLSTT on page 533
	QRYBLK on page 555
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBOVR on page 593
	RDBRLLBCK on page 598
	RDBUPDRM on page 604
	REBIND on page 606
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SCALAR on page 649
	SECCHK on page 652
	SECCHKRM on page 659
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLERRRM on page 710
	SQLSTT on page 714
	SUBSETS on page 747
	SYNCCRD on page 759
	SYNCCTL on page 760
	SYNCLOG on page 764
	SYNCPTMGR on page 782
	SYNCRRD on page 826
	SYNCRSY on page 828
	SYNTAXRM on page 835

Variable	Reference
	TCPSRCCD on page 853
	TCPSRCCR on page 856
	TCPSRCER on page 859
	TCPTRGER on page 861
	TRGNSPRM on page 868
	TYPDEF on page 872
	VALNSPRM on page 898
	OBJDSS on page 455
	RPYDSS on page 620
	RQSDSS on page 629

NAME

DSSFMT — Data Stream Structure Format

DESCRIPTION (Semantic)

Dictionary QDDBASD

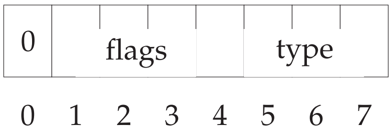
Codepoint X‘140D’

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

Bytes 1-3 are the flags. Bytes 4-7 are the type.

See the term CTLFMT for the description and format of the communications message DSS format byte.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
bit0	INSTANCE_OF LENGTH RESERVED	BITDR - A Single Bit 1 B’0’
bit1	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	BITDR - A Single Bit 1 B’0’ Structure not chained to next structure. B’1’ Structure chained to next structure.
bit2	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE NOTE	BITDR - A Single Bit 1 B’0’ Do not continue on error. B’1’ Continue on error. Must be B’0’ if the chaining bit (bit1) is

		B'0'.
bit3	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE NOTE	BITDR - A Single Bit 1 B'0' Next DSS has different request correlator. B'1' Next DSS has same request correlator. Must be B'0' if the chaining bit (bit1) is B'0'.
dsstyp	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE ENUVAL NOTE ENUVAL NOTE MINLVL ENUVAL NOTE MINLVL	HEXDR - Hexadecimal Number 1 1 Request DSS. 2 Reply DSS. 3 Object DSS. 4 Communications DSS. See the term CTLFMT. 4 5 Request DSS where no reply is expected. 5
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OBJDSS* on page 455
	RPYDSS on page 620
	RQSDSS on page 629
	SYNERRCD on page 831
	DSS on page 289
	SYNCPTOV on page 787

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

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

SEE ALSO

Variable	Reference
cmdrpy	*BNDSQLSTT* on page 130
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533

Variable	Reference
semantic	*BNDSQLSTT* on page 130
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533

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

Variable	Reference
semantic	*DDM* on page 255

ELMCLS

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'004D'	
value	INSTANCE_OF REQUIRED	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*ARRAY* on page 96

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 *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 *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 *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 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 3-34 on page 308 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) 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 the descriptions of the OPNQRY and QRYBLKCTL commands for more information about query protocols.

```
Assume the following command sequence has occurred:

BGNBND( RDBNAM(RDBxxxxxxxxxxxx)
        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)
        PKGDFTCC(CCSIDSBC(01F4) CCSIDDBC(012D)))

BNDSQLSTT( RDBNAM(RDBxxxxxxxxxxxx)
           PKGNAMCSN(RDBxxx,collid,package1,12345678,03)
           SQLSTTNBR(5))
SQLSTT( 'DECLARE CURSOR ...')
SQLSTTVRB(...)

BNDSQLSTT( RDBNAM(RDBxxxxxxxxxxxx)
           PKGNAMCSN(RDBxxx,collid,package1,12345678,05)
           SQLSTTNBR(7))
SQLSTT( 'UPDATE ...')
SQLSTTVRB(...)

ENDBND( RDBNAM(RDBxxxxxxxxxxxx)
        PKGNAMCT(RDBxxx,collid,package1,12345678)
        MAXSCTNBR(8))
```

Figure 3-34 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 *bndcrtctl* 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘2009’	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamct	INSTANCE_OF REQUIRED	PKGNAMCT - RDB Package Name and Consistency Token
maxsctnbr	INSTANCE_OF OPTIONAL	MAXSCTNBR - Maximum Section Number

clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	BNDSQLSTT on page 130
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGOWNID on page 512
	REBIND on page 606
	SQL on page 680
	SQLAM on page 694

ENDQRYRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'220B'	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL	SVRCOD - Severity Code 4 - WARNING - Warning Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475
semantic	*CLSQRY* on page 163
	CNTQRY on page 217
	FIXROWPRC on page 365

Variable	Reference
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYBLK on page 555
	QRYNOPRM on page 562
	SQLAM on page 694

ENDUOWRM

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.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'220C'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 4 - WARNING - Warning Severity Code
uowdsp	INSTANCE_OF REQUIRED	UOWDSP - Unit of Work Disposition
rdbnam	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336
	RDBCMM on page 582
	RDBRLLBCK on page 598
semantic	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336
	QRYNOPRM on page 562

Variable	Reference
	RDBCMM on page 582
	RDBRLLBCK on page 598

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘0040’	
value	INSTANCE_OF REQUIRED	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*ELMCLS* on page 306

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0015'	
value	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0016'	
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'.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*LVLCMP* on page 412
	QLFATT on page 554
	SUBSETS on page 747

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

Variable	Reference
insvar	*ABNUOWRM* on page 39
	BGNBNDRM on page 109
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	DSCINVRM on page 279
	DTAMCHRM on page 303
	ENDQRYRM on page 312
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	PKGBNARM on page 497

Variable	Reference
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RSCLMTRM on page 634
	SECCHKRM on page 659
	SQLERRRM on page 710
	SVRCOD on page 756
	SYNTAXRM on page 835
	TRGNSPRM on page 868
	VALNSPRM on page 898
semantic	*APPSRCER* on page 87
	APPTRGER on page 91
	BNDNERALW on page 126
	CMDCHKRM on page 170
	DSS on page 289
	SECCHKCD on page 656
	TCPTRGER on page 861

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 *Character Data Representation Architecture Registry* (SC09-1391, IBM).

SEE ALSO

Variable	Reference
insvar	*STTDATFMT* on page 741

EURTIMFMT

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 *Character Data Representation Architecture Registry* (SC09-1391, IBM).

SEE ALSO

Variable	Reference
insvar	*STTTIMFMT* on page 746
semantic	*STTTIMFMT* on page 746

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 the descriptions of SERVER and MANAGER).

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 code point dictionaries the other server has available. Each code point 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 code points 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 code points 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 code point 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 *srvrlslv* 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 *srvrlslv* 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 *srvnam* 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 *srvnam* 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*, *srvclsnm*, *srvnam*, and *srvrlslv* 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1041'	
extnam	INSTANCE_OF OPTIONAL IGNORABLE	EXTNAM - External Name

mgrlvlls	INSTANCE_OF OPTIONAL	MGRLVLLS - 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
srvrlslv	INSTANCE_OF OPTIONAL IGNORABLE	SRVRLSLV - Server Product Release Level
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
X'1443'	INSTANCE_OF REQUIRED	EXCSATRD - Server Attributes Reply Data
cmdrpy		**COMMAND REPLIES**
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	MGRLVLRM - 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

Variable	Reference
inscmd	*SUPERVISOR* on page 753
insvar	*DEPERRCD* on page 265
	MGRLVL on page 427
	MGRLVLRM on page 432
	PRCCNVCD on page 521
semantic	*ACCRDB* on page 42
	ACCSEC on page 51
	APPCMNI on page 68
	BNDOPTNM on page 128
	BNDOPTVL on page 129
	CCSIDMGR on page 140
	CMNAPPC on page 179
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	DCESECOVR on page 248
	EXCSATRD on page 329
	EXTENSIONS on page 346
	LVLCMP on page 412
	MGRLVLN on page 430
	MGRLVLRM on page 432
	SECMGR on page 663
	SUPERVISOR on page 753
	SYNCMNI on page 774
	SYNCPTOV on page 787
	TCPCMNI on page 840
	USRSECOVR on page 893

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 the descriptions of the terms EXCSAT, SERVER, and MANAGER)
- The target server's product release level
- The target server's external name
- The target server's name

DSS Carrier: OBJDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1443'	
extnam	INSTANCE_OF OPTIONAL	EXTNAM - External Name
mgrlvlls	INSTANCE_OF OPTIONAL	MGRLVLLS - Manager-Level List
srvclsnm	INSTANCE_OF OPTIONAL	SRVCLSNM - Server Class Name
srvnam	INSTANCE_OF OPTIONAL	SRVNAM - Server Name
srvrlslv	INSTANCE_OF OPTIONAL	SRVRLSLV - Server Product Release Level
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*EXCSAT* on page 323
semantic	*CCSIDMGR* on page 140
	DCESECOVR on page 248
	EXCSAT on page 323
	SYNCPTOV on page 787
	TCPCMNI on page 840
	USRSECOVR on page 893

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 *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 *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 *rdbcmtok* parameter specifies whether the RDB should allow the processing of commit or rollback operations.

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.

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.

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[12] unless *rdbcmtok* 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.

---

12. This reply message was added in DDM Level 4.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'200A'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
rdbcmtok	INSTANCE_OF OPTIONAL MINLVL DFTVAL	RDBCMTOK - RDB Commit Allowed 5 X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
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'2414'	INSTANCE_OF REQUIRED	SQLSTT - SQL Statement
rpydta		**REPLY OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4

	ENUVAL MINLVL OPTIONAL DFTVAL NOTE	'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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data

cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*APPSRCCD* on page 75
	BGNBND on page 101
	OPNQRY on page 475
	RDBALWUPD on page 580
	SQL on page 680
	SQLAM on page 694
	SYNCPTOV on page 787
	TCPSRCCD on page 853

EXCSQLSTT

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.

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.

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, its value must be the same as the *rdbnam* parameter specified on the ACCRDB command.

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 *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 the description of term OUTEXP for more explanation.

The *nbrrow* parameter specifies the number of rows to insert if the EXCSQLSTT's target operation is an insert or a multi-row insert.

The *prcnam* parameter specifies the stored procedure name. This name must follow the target system's semantic and syntactic rules for procedure names. See the term *PRCNAM* on page 525 for usage rules.

The *rdbcmtok* parameter specifies whether the RDB should allow the processing of commit or rollback operations.

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.

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.

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.

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:

RSLSETFLG
: Result Set Flags (*RSLSETFLG* on page 639)

PRCNAM
: Procedure Name (*PRCNAM* on page 525)

QRYBLKSZ
: Query Block Size (*QRYBLKSZ* on page 559)

MAXRSLCNT
: Maximum Result Set Count (*MAXRSLCNT* on page 423)

MAXBLKEXT
: Maximum Number of Extra Blocks (*MAXBLKEXT* on page 420)

When a procedure that generates result sets completes, an RSLSETRM, an SQLCARD or SQLDTARD, and an SQLRSLRD are returned, followed by at least an OPNQRYRM and the FD:OCA description of the data (QRYDSC) for each result set. An SQLCINRD may also be returned for each result set, depending on the setting of flags within the RSLSETFLG parameter. The block containing the end of the FD:OCA description may be completed, if room exists, with answer set data. Additional blocks of answer set data may also be chained to the block containing the end of the FD:OCA description, up to the maximum number of extra blocks of answer set data per result set specified in the MAXBLKEXT 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 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[13] unless *rdbcmtok* 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.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'200B'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name

13. This reply message was added in DDM Level 4.

pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
outexp	INSTANCE_OF OPTIONAL	OUTEXP - Output Expected
nbrrow	INSTANCE_OF OPTIONAL MINLVL	NBRROW - Number of Fetch or Insert Rows 4
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 qryblksz is REQUIRED if the EXCSQLSTT command invokes a stored procedure. 5
maxrslcnt	INSTANCE_OF OPTIONAL DFTVAL	MAXRSLCNT - Maximum Result Set Count 0
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks 0
rslsetflg	INSTANCE_OF OPTIONAL DFTVAL	RSLSETFLG - Result Set Flags B'00000000'
rdbcmtok	INSTANCE_OF OPTIONAL MINLVL DFTVAL	RDBCMTOK - RDB Commit Allowed 5 X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 ''

	NOTE	The default means the value received on the ACCRDB command is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides '' The default means the value received on the ACCRDB command is used.
X'2412'	INSTANCE_OF OPTIONAL NOTE	SQLDTA - SQL Program Variable Data Specified when the SQL statement has input variables.
rpydta		**REPLY OBJECTS**
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE REPEATABLE	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 REPEATABLE	TYPDEFOVR - TYPDEF Overrides '' The default means the value received on the ACCRDBRM is used.
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	QRYDSC - Query Answer Set Description

	NOTE	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 OPTIONAL May be returned for each result set generated by the invocation of a stored procedure.
cmdrpy		**COMMAND REPLIES**
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	CMMRQSRM - Commitment Request
	MINLVL	4
X'220A'	INSTANCE_OF	DSCINVRM - Invalid Description
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'220B'	INSTANCE_OF	ENDQRYRM - End of Query
	MINLVL	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	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'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

Variable	Reference
inscmd	*SQLAM* on page 694
insvar	*MAXRSLCNT* on page 423
	PRCNAM on page 525
	RSLSETFLG on page 639
semantic	*BGNBND* on page 101
	CNTQRY on page 217
	LMTBLKPRC on page 400
	MAXBLKEXT on page 420
	MAXRSLCNT on page 423
	OPNQRY on page 475
	OPNQRYRM on page 483
	PRPSQLSTT on page 533
	QRYBLK on page 555
	QRYBLKSZ on page 559
	QRYDTA on page 561
	RDBALWUPD on page 580
	RSLSETFLG on page 639

Variable	Reference
	RSLSETRM on page 642
	SQL on page 680
	SQLAM on page 694
	SQLCINRD on page 705
	SQLRSLRD on page 713
	STTASMEUI on page 740

EXPALL

NAME

EXPALL — Explain All Explainable SQL Statements

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'243B'

Length *

Class codpnt

The Explain All Explainable SQL Statements (EXPALL) specifies that the target SQLAM causes the target relational database (RDB) manager to explain normally any explainable SQL statements during the bind or rebind process.

SEE ALSO

Variable	Reference
insvar	*BNDEXPOPT* on page 123

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.

SEE ALSO

Variable	Reference
insvar	*BNDEXPOPT* on page 123
semantic	*ENDBND* on page 307

EXTENSIONS

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:

SUPERVISOR
Supervisor (*SUPERVISOR* on page 753)

SECMGR
Security Manager (*SECMGR* on page 663)

DICTIONARY
Dictionary (*DICTIONARY* on page 272)

CMNMGR
Communications Manager (*CMNMGR* on page 191)

AGENT Agent (*AGENT* on page 56)

CCSIDMGR
CCSID Manager (*CCSIDMGR* on page 140)

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 code points are assigned to uniquely identify each structure. The DDM architecture assigns these code points and does not allow duplicate definitions. However, for product-defined extensions to DDM, products must be able to assign code points 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

Variable	Reference
semantic	*CODPNTDR* on page 228
	CONCEPTS on page 237

EXTNAM

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'115E'	
value	INSTANCE_OF REQUIRED	NAMDR - Name Data
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
	EXCSATRD on page 329
semantic	*CCSIDMGR* on page 140
	EXCSAT on page 323

FALSE

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

Variable	Reference
insvar	*BOOLEAN* on page 136
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OUTEXP on page 489
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDBALWUPD on page 580
	RDBCMTOK on page 585
	RLSCONV on page 619
	RTNSQLDA on page 648
	SQLCSRHLD on page 706
	SYNCCTL on page 760
	TRGDFTRT on page 867
rpydta	*PRPSQLSTT* on page 533
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	BOOLEAN on page 136
	CNTQRY on page 217
	OUTEXP on page 489

Variable	Reference
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDBALWUPD on page 580
	RTNSQLDA on page 648
	SQLCSRHLD on page 706
	SYNCCTL on page 760
	TRGDFTRT on page 867

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 3-35. 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.

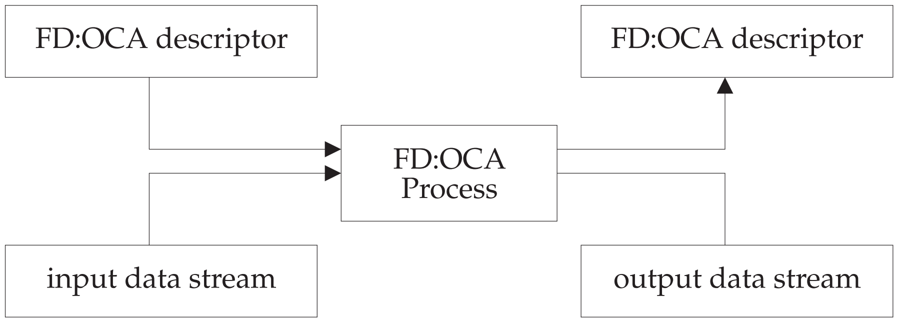


Figure 3-35 FD:OCA Processing Overview

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

Variable	Reference
insvar	*DSCERRCD* on page 277
semantic	*BNDSQLSTT* on page 130
	DSCERRCD on page 277
	DSCINVRM on page 279
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DTAMCHRM on page 303
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FDODSC on page 354
	FDODSCOFF on page 356
	FDODTA on page 357
	FDODTAOFF on page 359
	FDOOBJ on page 360

Variable	Reference
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362
	OPNQRY on page 475
	PRPSQLSTT on page 533
	QRYDSC on page 560
	SQL on page 680
	SQLAM on page 694
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLDARD on page 707
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714
	SQLSTTVRB on page 716
title	*FDODSC* on page 354
	FDODSCOFF on page 356
	FDODTA on page 357
	FDODTAOFF on page 359
	FDOOBJ on page 360
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362

FDODSC

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 the description of the term FD:OCA for a description of the FD:OCA triplets DDM uses.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0010'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DSCINVRM* on page 279
	FDOOBJ on page 360
	SQLDTA on page 708
	SQLDTARD on page 709
	TYPDEF on page 872
semantic	*EXCSQLSTT* on page 336
	FDODSCOFF on page 356
	FDODTA on page 357
	FDODTAOFF on page 359
	FDOOBJ on page 360
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362

Variable	Reference
	OPNQRY on page 475
	TYPDEF on page 872
sprcls	*QRYDSC* on page 560

FDODSCOFF

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'2118'	
value	INSTANCE_OF LENGTH MINVAL DFTVAL REQUIRED	BINDR - Binary Number Field 32 0 0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DSCINVRM* on page 279
semantic	*DSCINVRM* on page 279

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'147A'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*FDOOBJ* on page 360
	SQLDTA on page 708
	SQLDTARD on page 709
semantic	*EXCSQLSTT* on page 336
	FDODTAOFF on page 359
	FDOOBJ on page 360
	OPNQRY on page 475
sprcls	*QRYDTA* on page 561
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLDARD on page 707
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714

Variable	Reference
	SQLSTTVRB on page 716

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'2119'	
value	INSTANCE_OF LENGTH MINVAL DFTVAL REQUIRED	BINDR - Binary Number Field 32 0 0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

FDOOBJ

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 (INSMRWDSC) 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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1480'	
fdodsc	INSTANCE_OF REQUIRED REPEATABLE	FDODSC - FD:OCA Data Descriptor
fdodta	INSTANCE_OF OPTIONAL REPEATABLE	FDODTA - FD:OCA Data
insmrwdsc	INSTANCE_OF REQUIRED REPEATABLE	INSMRWDSC
insmrwdta	INSTANCE_OF REQUIRED REPEATABLE	INSMRWDTA
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
sprcls	*SQLDTA* on page 708
	SQLDTARD on page 709

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'212B'	
value	INSTANCE_OF LENGTH MINVAL REQUIRED	BINDR - Binary Number Field 16 0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DSCINVRM* on page 279
semantic	*DSCINVRM* on page 279

FDOTRPOFF

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'212A'	
value	INSTANCE_OF LENGTH MINVAL REQUIRED	BINDR - Binary Number Field 32 0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DSCINVRM* on page 279
semantic	*DSCINVRM* on page 279

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.

clsvar		**CLASS VARIABLES**
vldattls	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.
insvar	**NIL**	
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*CNSVAL* on page 216
	DFTVAL on page 269
	ENUVAL on page 318
	MAXVAL on page 425
	MINVAL on page 441
	RESERVED on page 617
	SPCVAL on page 676
semantic	*OBJECT* on page 459
sprcls	*BINDR* on page 112

Variable	Reference
	BITDR on page 115
	BITSTRDR on page 117
	BYTDR on page 137
	CHRDR on page 145
	CHRSTRDR on page 146
	CODPNTDR on page 228
	HEXDR on page 374
	HEXSTRDR on page 376
	NAMDR on page 444
	NAMSYMDR on page 447
	PKGCNSTKN on page 500
	PKGID on page 504
	PKGSN on page 519
	RDBCOLID on page 586

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) is for single row fetches (scrollable and non-scrollable cursors) and multi-row fetches (scrollable and non-scrollable cursors).

The fixed row query protocols are used for queries that have an ambiguous database 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 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 value is FRCFIXROW.

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). 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 is:

- An OPNQRYRM is returned.
- One or more QRYDSC reply data objects are returned.
- The query is suspended.

No QRYDTA reply data objects are returned in response to the OPNQRY command even though there might be space for them in the last query block.

The normal response to a CNTQRY command is:

- One or more QRYDTA reply data objects are returned.
- The query is suspended.

Each QRYDTA reply data object contains one or more answer set rows with associated SQLCAs or portions thereof. Only as many query blocks (see the description of QRYBLK for the definition of a query block) of QRYDTA reply data objects are returned since it is necessary for the target SQLAM to return all complete answer set rows that are requested. The QRYDTA reply data object in the first query block that a CNTQRY command returns must contain the first column or a partial column of an answer set row. The QRYDTA reply data object in the last or only query block a CNTQRY command returns must end with the last column of the last answer set row even though space can remain in the last query block.

The target SQLAM must use the block size specified on the OPNQRY or CNTQRY commands. This causes the target SQLAM to return multiple QRYDSC objects if the entire answer set description cannot fit in the specified block size or the target SQLAM can return multiple QRYDTA objects if all answer set rows cannot fit in the specified block size. The target SQLAM cannot return a query block that is larger than the specified block size. All query blocks the target SQLAM returns, except the last query block returned for each OPNQRY and CNTQRY command, must be of the specified size.

In the following sections, everything that appears on a line preceded by <== represents a single query block, 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.

Protocols for the OPNQRY Command

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 one or more QRYDSC reply data objects that contain the description of the answer set data.

```
OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)

           --- or ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) SQLCARD(--) QRYDSC(--)

           --- or ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)
                    <== QRYDSC(--)
                    <== QRYDSC(--)
```

- 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(--)
```

- For normal and non-terminating error (warning) conditions, an OPNQRYRM is returned followed by one or more QRYDSC reply data objects that describe the answer set data, and possibly an SQLCARD object.

```
OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)

           ---  or  ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) TYPDEFNAM(--)
                        TYPDEFOVR(--) SQLCARD(--) QRYDSC(--)
                    <== QRYDSC(--)
```

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, one or more QRYDTA reply data objects are returned that contain an SQLCA and a single row of answer set data for single row fetches, and for multi-row fetches, multiple answer set rows with associated SQLCAs.

```
CNTQRY(--) ==>
                    <== QRYDTA(--)

           ---  or  ---

CNTQRY(--) ==>
                    <== QRYDTA(--)
                    <== QRYDTA(--)
                    <== QRYDTA(--)
```

- For non-scrollable cursors, if a previous CNTQRY command returns the last row of answer set data, and the query is not terminated, then an ENDQRYRM and an SQLCARD object are returned. This terminates the query.

```
CNTQRY(--) ==>
                    <== ENDQRYRM(--) TYPDEFNAM(--)
                        TYPDEFOVR(--) SQLCARD(--)
```

- For multi-row fetches, if the target SQLAM is aware that the last row of answer set data is being returned, and space is available in the last query block for an ENDQRYRM and an SQLCARD object, then an ENDQRYRM and an SQLCARD object are placed in the last query block following the last QRYDTA reply data object. This terminates the query.

```
CNTQRY(--) ==>
                    <== QRYDTA(--) ENDQRYRM(--) SQLCARD(--)

           ---  or  ---

CNTQRY(--) ==>
                    <== QRYDTA(--)
                    <== QRYDTA(--)
                    <== QRYDTA(--) ENDQRYRM(--) SQLCARD(--)
```

- For multi-row fetches, if the target SQLAM is aware that the last row of the answer set data is being returned, and not enough space is available in the last query block for an ENDQRYRM and an SQLCARD object, then the response is the same as for normal conditions. The ENDQRYRM and the SQLCARD object are saved and are returned by the next CNTQRY command.

- 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(--) ==>
                    <== ABNUOWRM(--) 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

Variable	Reference
insvar	*QRYBLKCTL* on page 557
	QRYPRCTYP on page 566
semantic	*CNTQRY* on page 217
	ENDBND on page 307
	QRYBLK on page 555
	QRYBLKCTL on page 557
	SQLAM on page 694

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'1186'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State X'F1' - SYNCCRD SYNCTYPE(FORGET) requested. X'F0' - FALSE - False State X'F0' - Next reply implies forget. X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

FRCFIXROW

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) code point 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

Variable	Reference
insvar	*OPNQRY* on page 475
	QRYBLKCTL on page 557
semantic	*FIXROWPRC* on page 365
	LMTBLKPRC on page 400
	QRYBLKCTL on page 557

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0019'	
title	INSTANCE_OF	TITLE - Title
status	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
semantic	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 For each line of help text. description
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
class	*ABBREVIATIONS* on page 30
	AGNCMDPR on page 60
	AGNRPYPR on page 63
	APPCMNFL on page 64
	APPCMNI on page 68
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CMNLYR on page 189

Variable	Reference
	CMNOVR on page 196
	CONCEPTS on page 237
	DCESECOVR on page 248
	DCESECTKN on page 252
	DDM on page 255
	DTAOVR on page 305
	EXTENSIONS on page 346
	FDOCA on page 351
	INHERITANCE on page 380
	LVLCMP on page 412
	MGROVR on page 436
	NWPWDSEC on page 454
	OBJOVR on page 464
	OSFDCE on page 488
	PRCOVR on page 526
	PWDSEC on page 537
	QRYBLK on page 555
	RDBOVR on page 593
	RESYNOVR on page 618
	SECOVR on page 667
	SNASECOVR on page 675
	SQL on page 680
	SRVOVR on page 728
	STRLYR on page 737
	SUBSETS on page 747
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCMNI on page 774
	SYNCMNT on page 778
	SYNCPTOV on page 787
	TCPCMNFL on page 838
	TCPCMNI on page 840

Variable	Reference
	TCPCMNT on page 844
	TCPIPOVR on page 847
	TCPSRCCD on page 853
	TCPSRCCR on page 856
	TCPSRCER on page 859
	TCPTRGER on page 861
	USRIDSEC on page 892
	USRSECOVR on page 893
semantic	*CLASS* on page 148
	DICTIONARY on page 272
	INHERITANCE on page 380
sprcls	*CODPNT* on page 225
	CONSTANT on page 238
vldattls	*AGENT* on page 56
	CCSIDMGR on page 140
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	DICTIONARY on page 272
	MANAGER on page 417
	RDB on page 571
	SECMGR on page 663
	SERVER on page 670
	SQLAM on page 694
	SUPERVISOR on page 753
	SYNCPTMGR on page 782

HEXDR

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'.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF LENGTH REQUIRED	BITSTRDR - Bit String Field 4
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*CNSVAL* on page 216
	CODPNTDR on page 228
	DFTVAL on page 269
	DSSFMT on page 301
	ENUVAL on page 318
	HEXSTRDR on page 376
	MAXVAL on page 425
	MINVAL on page 441

Variable	Reference
	RESERVED on page 617
	SPCVAL on page 676
semantic	*HEXSTRDR* on page 376
	INHERITANCE on page 380

HEXSTRDR

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'.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF REPEATABLE REQUIRED	HEXDR - Hexadecimal Number
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BOOLEAN* on page 136
	CCSIDDBC on page 138
	CCSIDMBC on page 139
	CCSIDSBC on page 144
	CMMTYP on page 178
	CNSVAL on page 216
	CODPNTDR on page 228
	DEPERRCD on page 265
	DFTVAL on page 269

Variable	Reference
	ENUVAL on page 318
	MAXVAL on page 425
	MINVAL on page 441
	OUTEXP on page 489
	PRCCNVCD on page 521
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	RDBALWUPD on page 580
	RDBCMTOK on page 585
	RESERVED on page 617
	RLSCONV on page 619
	RTNSQLDA on page 648
	SECCHKCD on page 656
	SPCVAL on page 676
	SQLCSRHLD on page 706
	SYNCCTL on page 760
	SYNERRCD on page 831
	TRGDFTRT on page 867
	UOWDSP on page 882
semantic	*INHERITANCE* on page 380
	DSSFMT on page 301

IGNORABLE

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'001C'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	OPNQRYRM on page 483
	RDBUPDRM on page 604
	RSLSETRM on page 642
	SECCHKRM on page 659
	SVRCOD on page 756
semantic	*SECCHK* on page 652
	SECCHKCD on page 656
	USRSECOVR on page 893

INHERITANCE

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 3-36 which displays the inheritance of a zoological taxonomy.

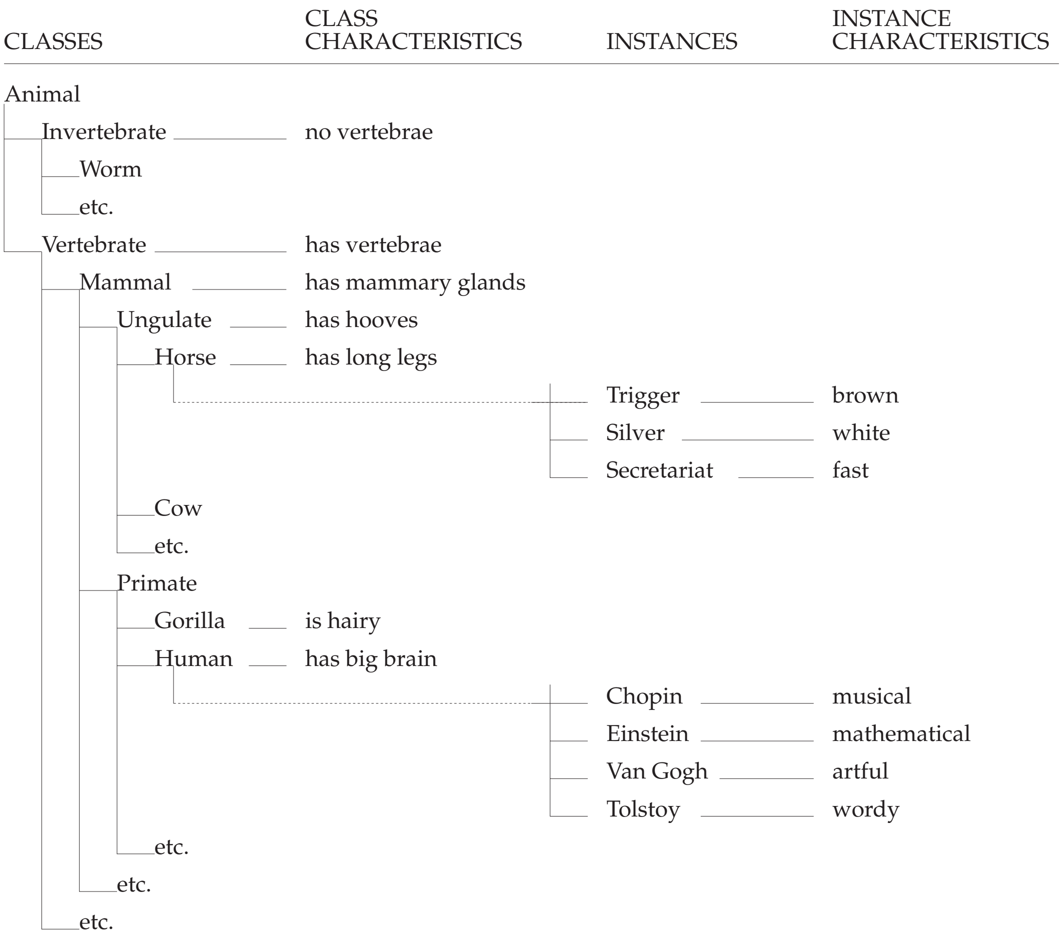


Figure 3-36 Inheritance in a Zoological Taxonomy

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 3-37. 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.

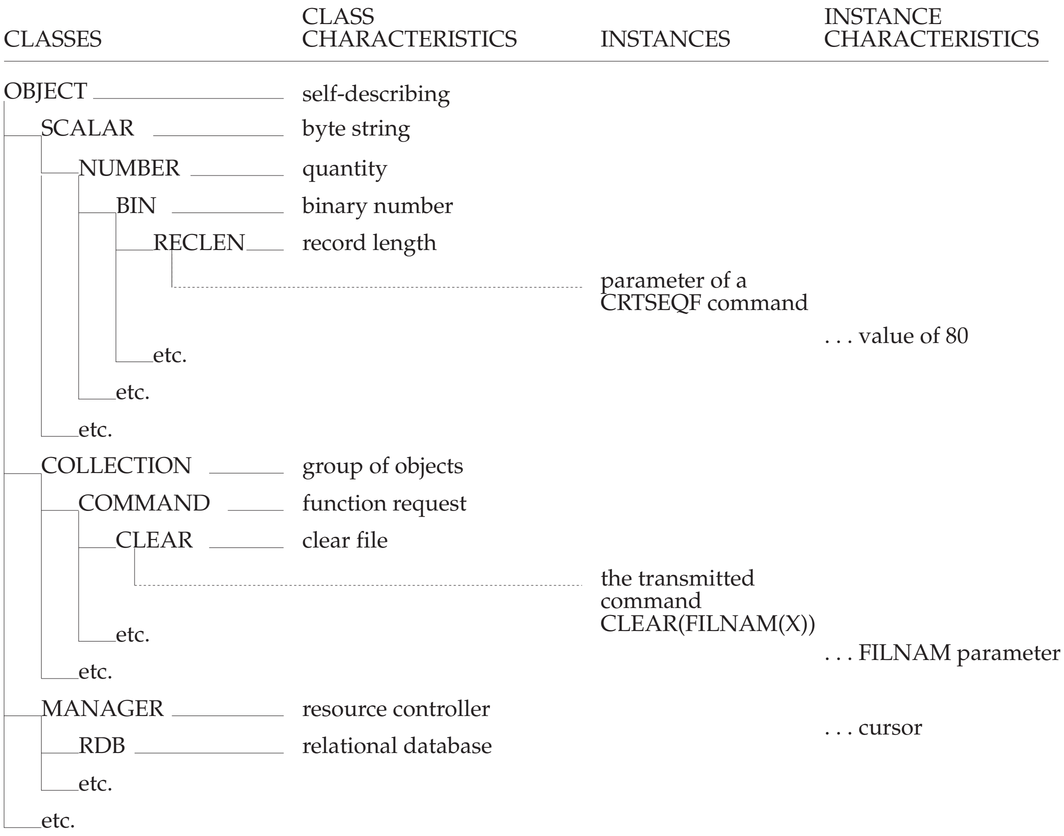


Figure 3-37 Inheritance in the DDM Taxonomy

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 3-38 on page 382 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.

DATA . . .	Encoded Data Classes
OBJECT . . .	Data Object Classes
MANAGER . . .	Object Management Classes
SERVER . . .	Manager Control Classes

Figure 3-38 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 3-39 define the encoding scheme.

Encoded Data Classes

- DATA
  - BINDR . . .
  - BITDR
  - BITSTRDR . . .
    - CHRDR . . .
      - LETTER
      - DGTDR . . .
      - NAMDR . . .
  - BYTDR
  - HEXDR
  - HEXSTRDR . . .
  - CHRSTRDR . . .
  - DGTSTRDR
  - HELP . . .
  - OBJDSS
  - RPYDSS
  - RQSDSS
  - CMNDSS
  - OBJECT . . .

Figure 3-39 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 3-40 on page 383). 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.

Data Object Level

- OBJECT . . .
  - SCALAR . . .
    - MAGNITUDE
      - DATE . . .
      - NUMBER
        - BIN
        - BOOLEAN . . .
      - STRING . . .
        - NAME . . .
        - RECORD . . .
  - COLLECTION . . .
    - ASSOCIATION . . .
      - ATTRIBUTE . . .
    - CLASS . . .
    - COMMAND . . .
    - RPYMSG . . .
    - CMNMSG . . .
  - MANAGER . . .

Figure 3-40 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 3-41).

Object Management Level

- MANAGER
  - SUPERVISOR
  - SECMGR
  - RDB
  - SQLAM
  - SYNCPTMGR
  - SYSCMDMGR
  - DICTIONARY . . .
  - AGENT
  - CMNMGR
    - CMNAPPC
    - CMNMLT
    - CMNSYNCPT
  - SERVER . . .

Figure 3-41 Object Management Classes Hierarchy

The management control classes of DDM are managers of managers and therefore, implement data management servers (see Figure 3-42 on page 384).

SERVER . . .

Figure 3-42 Manager Control Hierarchy

Note: The architects manually implemented and verified the inheritance of structure and characteristics to enhance this DDM Reference.

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
semantic	*CLASS* on page 148
	CONCEPTS on page 237

INHERITED

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 code point 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 *lsvar* of a CLASS only specifies the INHERITED attribute. For example, the *lsvar* of this CLASS specifies only the INHERITED attribute because only the variables are inherited from the superclasses of this CLASS.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0049'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
semantic	*CLASS* on page 148

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 code point of a class object as its value. It indicates that the value of the variable described is an instance of the specified class.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'005D'	
value	INSTANCE_OF NOTE REQUIRED	CODPNTDR - Code Point Data Representation Must be the code point of a class object.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*ATTLST* on page 99
	DEFLST on page 263
	INHERITANCE on page 380
	LVLCMP on page 412

IPADDR

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 4 bytes of IPADDR is the binary IP address followed by 2 bytes binary TCP port number.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SRVLSRV* on page 723
	SYNCLOG on page 764
semantic	*SYNCPTOV* on page 787

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:

```
yyy-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

Variable	Reference
insvar	*STTDATFMT* on page 741
semantic	*STTDATFMT* on page 741

ISOLVLALL

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

Variable	Reference
insvar	*PKGISOLVL* on page 505
semantic	*PKGISOLVL* on page 505

ISOLVLCHG

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

Variable	Reference
insvar	*PKGISOLVL* on page 505
semantic	*PKGISOLVL* on page 505

ISOLVLCS

NAME

ISOLVLCS — Isolation Level Cursor Stability

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2442'

Length *

Class codpnt

Isolation Level Cursor Stability (ISOLVLCS) 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

Variable	Reference
insvar	*PKGISOLVL* on page 505
semantic	*PKGISOLVL* on page 505

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

Variable	Reference
insvar	*PKGISOLVL* on page 505
semantic	*PKGISOLVL* on page 505

ISOLVLRR

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

Variable	Reference
insvar	*PKGISOLVL* on page 505
semantic	*PKGISOLVL* on page 505

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 *Character Data Representation Architecture Registry* (SC09-1391, IBM).

SEE ALSO

Variable	Reference
insvar	*STTTIMFMT* on page 746
semantic	*STTTIMFMT* on page 746

JISDATFMT

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

Variable	Reference
insvar	*STTDATFMT* on page 741
semantic	*STTDATFMT* on page 741

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 *Character Data Representation Architecture Registry* (SC09-1391, IBM).

SEE ALSO

Variable	Reference
insvar	*STTTIMFMT* on page 746

LENGTH

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'001F'	
value	INSTANCE_OF	BINDR - Binary Number Field
	ENULEN	16
	ENULEN	32
	ENULEN	48
	ENULEN	64
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BIN* on page 110
semantic	*APPCMNFL* on page 64
	APPCMNI on page 68
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	ARRAY on page 96

Variable	Reference
	ATTLST on page 99
	BINDR on page 112
	COMMAND on page 233
	DEFLST on page 263
	MGRLVL on page 427
	OBJECT on page 459
	OBJOVR on page 464
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCMNI on page 774
	SYNCMNT on page 778

LMTBLKPRC

NAME

LMTBLKPRC — Limited Block Query 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.

Limited block query protocols are used for queries with ambiguous database cursors if both of the following are true:

- The OPNQRY command did not specify FRCFIXROW in the *qryblkctl* parameter.
- The package *qryblkctl* parameter value is LMTBLKCTL.

Limited block query protocols are used for queries with a non-ambiguous database cursor if all of the following are true:

- The database cursor is declared FETCH ONLY, or 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 value is not FRCFIXROW.

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). 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 normal response to an OPNQRY command is:

- An OPNQRYRM is returned.
- One or more query answer set description QRYDSC reply data objects are returned.
- Possibly, a single query answer set data QRYDTA reply data object is returned if space is left in the query block containing the last QRYDSC reply data object. See the description of the term QRYBLK for a definition of a query block.
- Possibly, additional query blocks, limited in number by the value of the MAXBLKEXT parameter of OPNQRY, each containing a QRYDTA reply data object.
- The query is suspended.

The normal response to a CNTQRY command is:

- One or more QRYDTA reply data objects containing the end of a row are returned.
- Possibly, additional answer set data filling the first query block that contains the end of a row.
- Possibly, additional query blocks, limited in number by the value of the MAXBLKEXT parameter of CNTQRY, each containing a QRYDTA reply data object.

- The query or result set is suspended.

Each QRYDTA object can contain one or more either partial or complete SQLCAs and answer set data rows.

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.

An EXCSQLSTT command may invoke a stored procedure that returns one or more result sets. After the EXCSQLSTT or each subsequent CNTQRY command, each result set is suspended unless some condition terminates the result set (see EXCSQLSTT and CNTQRY). 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 any time the CLSQRY command suspends it.

The normal response to an EXCSQLSTT command that invokes a stored procedure that generates result sets is:

- A summary component.

  The summary component consists of:

  — An RSLSETRM with a PKGSNLST reply data parameter containing a PKGNAMCSN entry for each result set.

  — An SQLCARD or SQLDTARD containing general status information about the success or failure of the execution of the stored procedure. If the invocation of the stored procedure requires the specification of input host variables or the stored procedure returns output parameter values, then an SQLDTARD is used. Otherwise, an SQLCARD is used.

  — An SQLRSLRD reply data object containing additional application-specific information about each query result set.

- At most M query result set components, where M is the MAXRSLCNT parameter value of EXCSQLSTT. The query result set components follow the summary component.

  Each query result set component consists of:

  — An OPNQRYRM.

  — Possibly, an SQLCINRD reply data object containing information about the columns within the result set.

  — One or more query answer set description QRYDSC reply data objects.

  — Possibly, a single query answer set data QRYDTA reply data object if space is left in the query block containing the last QRYDSC reply data object. See the description of the term QRYBLK for a definition of a query block.

  — Possibly, additional query blocks, limited in number by the value of the MAXBLKEXT parameter of EXCSQLSTT, each containing a QRYDTA reply data object.

- Each result set is suspended.

The target SQL access manager (SQLAM) must use the query block size specified on the OPNQRY, CNTQRY, or EXCSQLSTT commands. The target SQLAM cannot return a query block that is larger than the specified query block size. All query blocks the target SQLAM returns for the OPNQRY or CNTQRY command except possibly the last query block must be of the specified query block size. All query blocks the target SQLAM returns for the EXCSQLSTT command except possibly the blocks of the summary component, a block preceding a block that contains an OPNQRYRM, or the last query block must be of the specified query block size.

Options for Using Space Within the Last QRYBLK

The options for using the space within the last query block in response to an OPNQRY or a CNTQRY command or the last query block of a query result set component in response to an EXCSQLSTT command are based on the following conditions. The examples in Figure 3-43 illustrate these conditions and options.

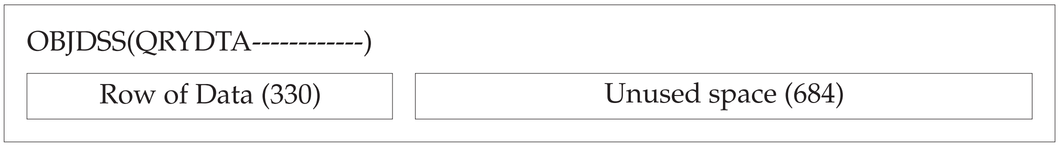
- CONDITIONS
  - — A row (or the end of a row) has been placed in the query block.
  - — The query block is not full.
- OPTIONS
  1. The remaining space in the last query block is not used. It is sent as a short query block.
  2. The remaining space in the query block is used for additional complete rows (subsequent rows of data fitting completely within the remaining space in the query block). When the last query block is not filled completely, it is sent as a short query block.
  3. The remaining space in the query block is used completely; the last query block is filled completely with row data. The last row of data might not fit in the block. As much of the data is placed in the block as is needed to fill it. This row of data is a partial row. The last query block is sent as a full query block at the specified query block size. If the block ended with a partial row of data, then the remainder of the row is sent in response to the next CNTQRY command.

Figure 3-43 Examples of Space Utilization in the Last Query Block

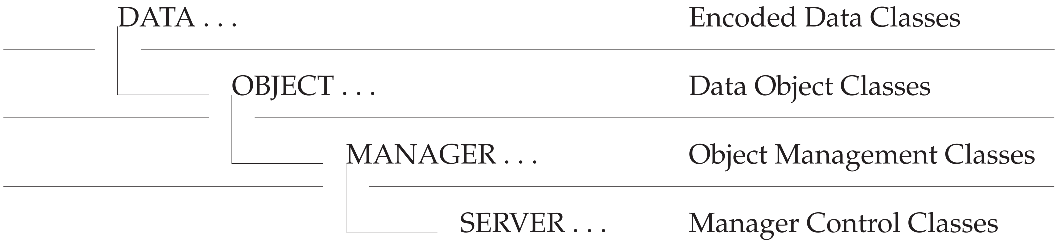
Assume:

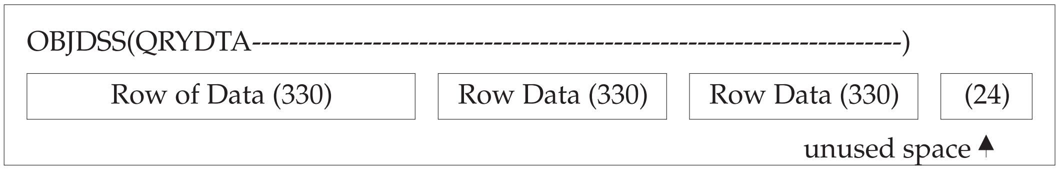
OBJDSS is 6 bytes
QRYDTA is 4 bytes
One row is 330 bytes
The QRYBLK is 1024 bytes

1. The empty space is not used.



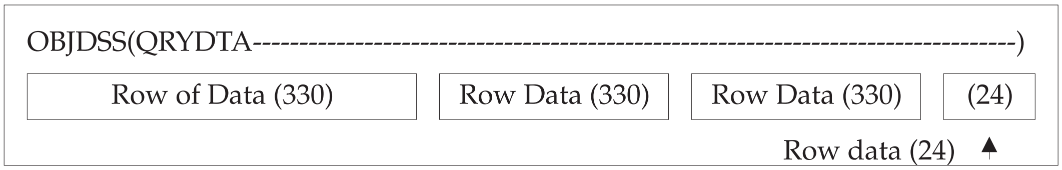
SEND short QRYBLK(340)





SEND short QRYBLK(1000)

3. The empty space is used completely.



SEND full QRYBLK(1024)

Options for Using Space Within the Summary Component

The options for using space within the Summary Component of the response to an EXCSQLSTT command are based on the following conditions:

- CONDITIONS
  - — The RSLSETRM has been placed in the query block or the RSLSETRM and SQLCARD or RSLSETRM and SQLDTARD have been placed in the query block.
  - — The query block is not full, but fewer than 4 unused bytes remain.
- OPTIONS
  1. The remaining space in the query block is not used. It is sent as a short query block.

Figure 3-44 Example of Space Utilization in the Summary Component

Assume:

RPYDSS is 6 bytes
RSLSETRM is 4 bytes
The QRYBLK is 566 bytes

1. The empty space is not used.

RPYDSS(RSLSETRM)

SVRCOD (6)	PKGSNLST (548)	(2)

unused space (2)

SEND short QRYBLK(564)

- CONDITIONS
  - — The RSLSETRM has been placed in the query block or the RSLSETRM and SQLCARD or RSLSETRM and SQLDTARD have been placed in the query block.
  - — The query block is not full and at least 4 unused bytes remain, but there are not enough remaining unused bytes for the entire reply data object (SQLCARD, SQLDTARD, or SQLRSLRD) that follows.
- OPTIONS
  1. The remaining space within the query block is used completely.

Figure 3-45 Example of Space Utilization in the Summary Component

Assume:

RPYDSS is 6 bytes
RSLSETRM is 4 bytes
OBJDSS is 6 bytes
The QRYBLK is 574 bytes

1. The empty space is used completely.

RPYDSS(RSLSETRM) OBJDSS(SQLCARD)

SVRCOD (6)	PKGSNLST (548)	(4)

SQLCARD LL+CP (4)

SEND full QRYBLK(574)

- CONDITIONS
  - — The SQLRSLRD has been placed in the query block.
  - — The query block is not full.
- OPTIONS

1. The remaining space in the last query block is not used. It is sent as a short query block.

Figure 3-46 Example of Space Utilization in the Summary Component

Assume:

OBJDSS is 6 bytes
SQLRSLRD is 90 bytes
The QRYBLK is 1024 bytes

1. The empty space is not used.

OBJDSS(SQLRSLRD)	
SQLRSLRD (90)	Unused space (928)

SEND short QRYBLK(96)

Protocol Examples

In the following protocol lists, everything that appears on a line preceded by <== represents a single query block and (--) represents the parameter list.

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

The following examples show some of the valid responses to the OPNQRY command:

- For normal and non-terminating error conditions, an OPNQRYRM is returned preceding one or more QRYDSC reply data objects and possibly a single QRYDTA object if space is left in the last query block.

```
OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)

           ---  or  ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) SQLCARD(--) QRYDSC(--)

           ---  or  ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--) QRYDTA(--)

           ---  or  ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)
                    <== QRYDSC(--)
                    <== QRYDSC(--)

           ---  or  ---
```

```
OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--)
                    <== QRYDSC(--)
                    <== 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 target SQLAM is aware that the last of the answer set data is being returned and if enough space is in the last query block for an ENDQRYRM and an SQLCARD object, then an ENDQRYRM and an SQLCARD object are placed in the last query block following the last QRYDTA reply data object. This terminates the query.

```
OPNQRY(--) ==>
                    <== OPNQRYRM(--) QRYDSC(--) QRYDTA(--)
                    <== ENDQRYRM(--) SQLCARD(--)

           --- or ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) SQLCARD(--) QRYDSC(--)
                    <== QRYDSC(--) QRYDTA(--) ENDQRYRM(--)
                    <== SQLCARD(--)

           --- or ---

OPNQRY(--) ==>
                    <== OPNQRYRM(--) TYPDEFNAM(--) TYPDEFOVR(--)
                    <== QRYDSC(--) QRYDTA(--)
                    <== ENDQRYRM(--) TYPDEFNAM(--) TYPDEFOVR(--)
                    <== SQLCARD(--)
```

- If the target SQLAM is aware that the last of the answer set data is being returned and if enough space is not in the last query block for an ENDQRYRM and an SQLCARD object, then the response is the same as for normal conditions. The ENDQRYRM and the SQLCARD object are saved and are returned by the next CNTQRY command.
- 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, and if space is available in the query block for an ENDQRYRM and an SQLCARD object, 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(--)
```

- If the target RDB detects an error condition preventing the database cursor from remaining open after the OPNQRYRM is returned and if space is not available in the query block for an ENDQRYRM and an SQLCARD object, then the ENDQRYRM and the SQLCARD object are saved and are returned by the next CNTQRY command.
- 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

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 are returned. The last or only query block returned is the first query block that contains the last column of a row in the answer set data.

```
CNTQRY(--) ==>
                    <== QRYDTA(--)

           ---  or  ---

CNTQRY(--) ==>
                    <== QRYDTA(--)
                    <== QRYDTA(--)
                    <== QRYDTA(--)
```

- If the OPNQRY or CNTQRY command saves the reply message and an SQLCARD object, then the saved reply message and SQLCARD object are returned. This terminates the query.

```
CNTQRY(--) ==>
                    <== ENDQRYRM(--) SQLCARD(--)
```

- If the target SQLAM is aware that the last of the answer set data is being returned and enough space is in the last query block for an ENDQRYRM and an SQLCARD object, then an ENDQRYRM and an SQLCARD object are placed in the last query block following the last QRYDTA reply data object. This terminates the query.

```
CNTQRY(--) ==>
                    <== QRYDTA(--) ENDQRYRM(--) SQLCARD(--)

           ---  or  ---

CNTQRY(--) ==>
                    <== QRYDTA(--)
                    <== QRYDTA(--)
                    <== QRYDTA(--) ENDQRYRM(--) SQLCARD(--)
```

- If the target SQLAM is aware that the last of the answer set data is being returned and if enough space is not in the last query block for an ENDQRYRM and an SQLCARD object, then the response is the same as for normal conditions. The ENDQRYRM and the SQLCARD object are saved and are returned by the next CNTQRY command.

- If a OPNQRY or CNTQRY command returns the last row of answer set data and the query is not terminated, then an ENDQRYRM and an SQLCARD object are returned. This terminates the query.

```
CNTQRY(--) ==>
                    <== ENDQRYRM(--) SQLCARD(--)
           ---  or  ---
CNTQRY(--) ==>
                    <== ENDQRYRM(--) TYPDEFNAM(--)
                        TYPDEFOVR(--) 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.

```
CLSQRY(--) ==>
                    <== 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.

```
CLSQRY(--) ==>
                    <== SQLCARD(--)
```

- For terminating error conditions that prevent the CLSQRY command from being passed to the target SQLAM, an appropriate reply message (RDB not accessed reply message, in the example below) is returned.

```
CLSQRY(--) ==>
                    <== RDBNACRM(--)
```

SEE ALSO

Variable	Reference
insvar	*QRYBLKCTL* on page 557
	QRYPRCTYP on page 566
rpydta	*OPNQRY* on page 475
semantic	*CNTQRY* on page 217
	OPNQRY on page 475
	QRYBLK on page 555
	QRYBLKCTL on page 557
	SQLAM on page 694

LOGNAME

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	22	
class	X'1184'	
value	INSTANCE_OF LENGTH	BYTSTRDR - Byte String 18
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCLOG* on page 764
semantic	*LOGTSTMP* on page 411

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	22	
class	X'1185'	
value	INSTANCE_OF LENGTH	CHRSTRDR - Character String 18
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCLOG* on page 764

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.[14]

     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.[15] New reply data objects can be returned for new conditions or situations.

---

14. 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.
15. 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 =< N =< 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 =< N =< 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

Variable	Reference
semantic	*CONCEPTS* on page 237

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.

clsvar		CLASS VARIABLES
mgrlvln	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 the term MGRLVLN for the rules to be followed in setting manager levels.
mgrdepls	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 MGRLVLN of a manager on which the manager being defined has a semantic or functional dependency. A NOTE describing the dependency should be provided. See the term SUBSETS for a description of the effects of manager level dependencies on product subset selection.
vldattls	INSTANCE_OF TITLE NOTE	DEFLST - Definition List Valid Attributes List The list of attributes that can be specified for a manager. The **vldattls** variable of the manager's class determines the attributes of the class.
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1456'	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	2	
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
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

Variable	Reference
	EXCSAT on page 323
	EXCSATRD on page 329
	INHERITANCE on page 380

Variable	Reference
	OBJECT on page 459
	SERVER on page 670
	STRLYR on page 737
sprcls	*AGENT* on page 56
	CCSIDMGR on page 140
	CMNMGR on page 191
	DICTIONARY on page 272
	RDB on page 571
	SECMGR on page 663
	SERVER on page 670
	SQLAM on page 694
	SUPERVISOR on page 753
	SYNCPTMGR on page 782

MAXBLKEXT

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.

clsvar	**NIL**	
insvar		**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
	DFTVAL	result set.
	OPTIONAL	0

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475

Variable	Reference
semantic	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	LMTBLKPRC on page 400
	OPNQRY on page 475
	RSLSETFLG on page 639

MAXLEN

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0021'	
value	INSTANCE_OF ENULEN ENULEN ENULEN ENULEN REQUIRED	BINDR - Binary Number Field 16 32 48 64
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2140'	
value	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

Variable	Reference
insvar	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336
	LMTBLKPRC on page 400

MAXSCTNBR

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2127'	
value	INSTANCE_OF LENGTH MINVAL DFTVAL NOTE REQUIRED	BINDR - Binary Number Field 16 1 '' The default value is the greater of one, or the highest section number a BNDSQLSTT statement references for the current bind process.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ENDBND* on page 307
semantic	*BGNBND* on page 101
	ENDBND on page 307

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0022'	
value	SPRCLS	FIELD - A Discrete Unit of Data
	NOTE	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1218'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
deperrcd	INSTANCE_OF REQUIRED	DEPERRCD - Manager Dependency Error Code
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	EXCSAT on page 323
	SECCHK on page 652
semantic	*ACCRDB* on page 42

NAME

MGRLVL — Manager Level

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1442'

Length *

Class CLASS

Sprcls DATA - Encoded Information

Manager Level (MGRLVL) associates the code point 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
codpnt	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	NOTE	Specifies the code point 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
	NOTE	This must be treated as an open-ended list because of its use in the EXCSAT command. That is, code points other than those in the previously enumerated value list must be accepted and handled as the EXCSAT command requires.
	REQUIRED	

mgrlvln	INSTANCE_OF SPCVAL NOTE REQUIRED	MGRLVLN - Manager-Level Number Attribute 0 The *mgrlvln* parameter documented in each manager term determines the manager level number.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*MGRLVLLS* on page 429
semantic	*DCESECOVR* on page 248
	LVLCMP on page 412
	SCALAR on page 649
	USRSECOVR on page 893

NAME

MGRLVLLS — Manager-Level List

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1404'

Length *

Class CLASS

Sprcls SCALAR - Scalar Object

Manager-Level List (MGRLVLLS) Scalar Object specifies a list of code points and support levels for the classes of managers a server supports.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1404'	
mgrlvl	INSTANCE_OF REPEATABLE REQUIRED	MGRLVL - Manager Level
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
	EXCSATRD on page 329
	MGRLVLRM on page 432
semantic	*CCSIDMGR* on page 140
	DCESECOVR on page 248
	EXCSAT on page 323
	LVLCMP on page 412
	MGRLVLRM on page 432
	USRSECOVR on page 893

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1473'	
value	INSTANCE_OF LENGTH REQUIRED	BINDR - Binary Number Field 16
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*MANAGER* on page 417
insvar	*MGRLVL* on page 427
semantic	*EXCSAT* on page 323

NAME

MGRLVLRM — Manager-Level Conflict

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1210'

Length *

Class CLASS

Sprcls RPYMSG - Reply Message

Manager-Level Conflict (MGRLVLRM) Reply Message indicates that the manager levels specified in the MGRLVLLS 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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1210'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
mgrlvla	INSTANCE_OF REQUIRED NOTE	MGRLVLLS - Manager-Level List The manager levels that the MGRLVLLS parameter specifies on EXCSAT conflict.
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSAT* on page 323
semantic	*EXCSAT* on page 323

Variable	Reference
	LVLCMP on page 412

MGRNAM

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1452'	
value	INSTANCE_OF MINLEN MAXLEN REQUIRED	BYTSTRDR - Byte String 0 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*AGNCMDPR* on page 60
sprcls	*PRCNAM* on page 525
	RDBNAM on page 589
	SECMGRNM on page 666
	SPVNAM on page 679
	SRVNAM on page 727
vldattls	*CCSIDMGR* on page 140
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197

Variable	Reference
	CMNTCPIP on page 209
	DICTIONARY on page 272
	MANAGER on page 417
	SQLAM on page 694
	SYNCPTMGR on page 782

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 3-47 on page 437. 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 3-48 on page 437 and Figure 3-49 on page 438. 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 (*CCSIDMGR* on page 140)

CMNOVR
Communications Overview (*CMNOVR* on page 196)

RDBOVR
Relational Database Overview (*RDBOVR* on page 593)

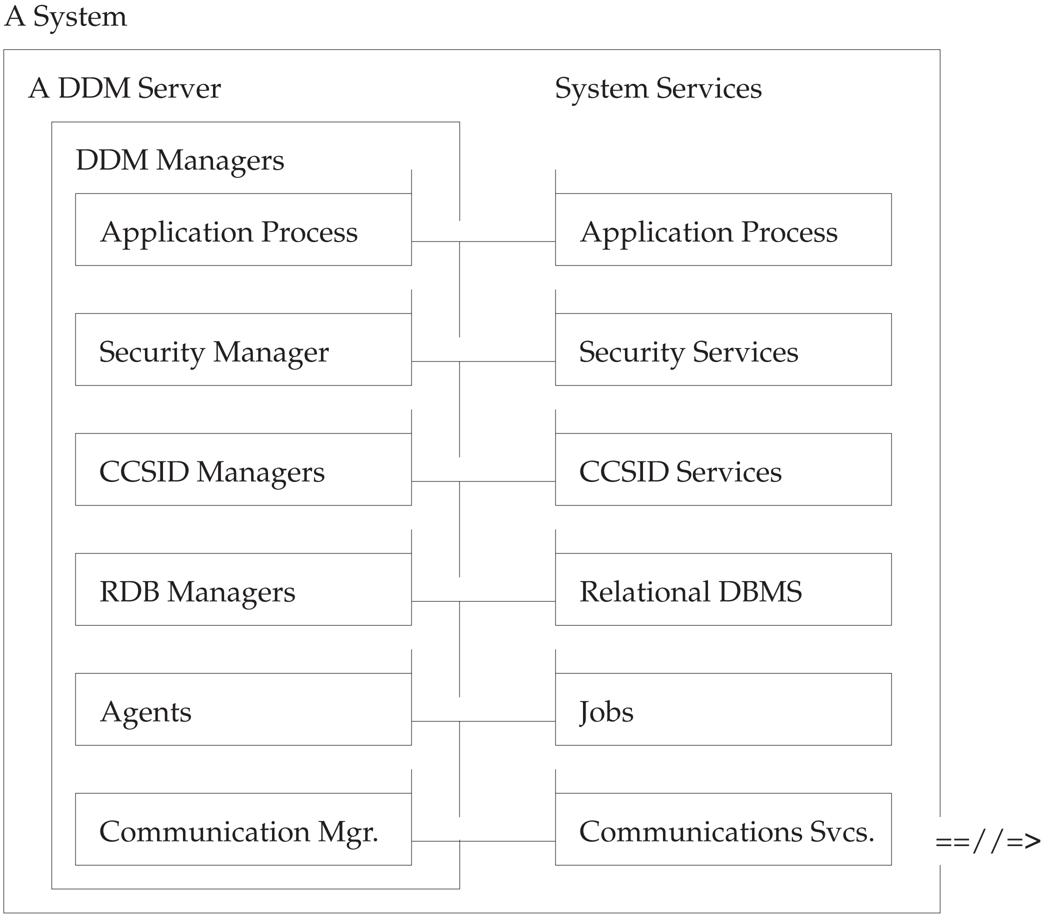


Figure 3-47 Mapping a DDM Server onto a System

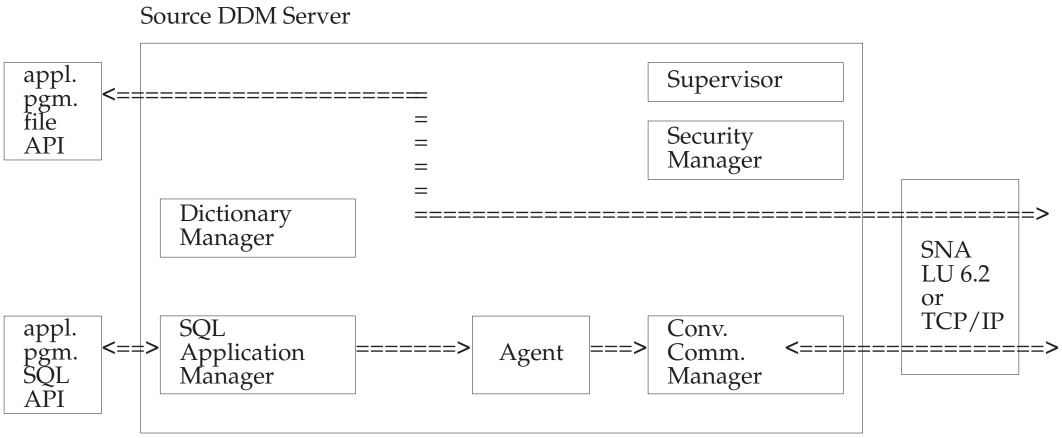


Figure 3-48 Source Server Manager Interactions

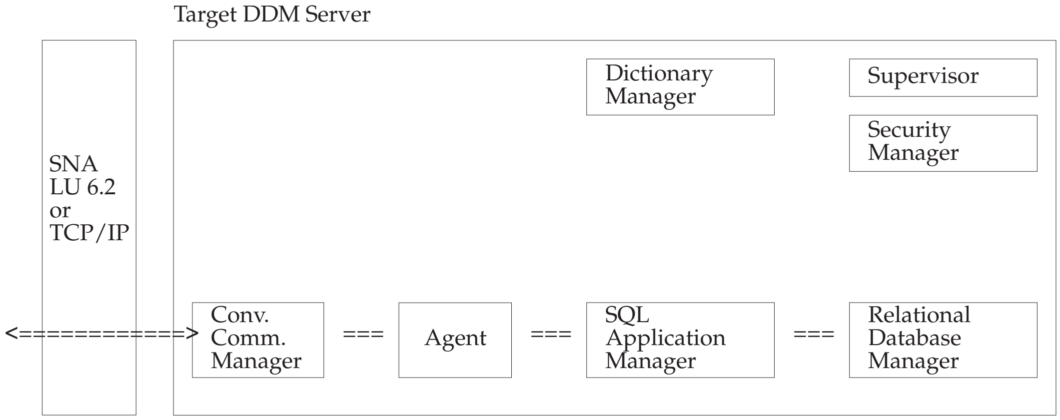


Figure 3-49 Target Server Manager Interactions

SEE ALSO

Variable	Reference
semantic	*DDM* on page 255

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0025'	
value	INSTANCE_OF ENULEN ENULEN ENULEN ENULEN REQUIRED	BINDR - Binary Number Field 16 32 48 64
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0002'	
value	INSTANCE_OF LENGTH REQUIRED	BINDR - Binary Number Field 16
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*LVLCMP* on page 412
	QLFATT on page 554

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0026'	
value	SPRCLS NOTE REQUIRED	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'.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

MTLEXC

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 code point an object that cannot exist in the same collection as the object with the MTLEXC attribute.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0067'	
value	INSTANCE_OF REQUIRED	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*LVLCMP* on page 412
	REQUIRED on page 615

NAME

MTLINC — Mutually Inclusive Attribute

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Codepoint 00A7

Length *

Class CLASS

Sprcls OBJECT - Self-Identifying Data

Mutually Inclusive Attribute (MTLINC) identifies by code point an object that must exist in the same collection as the object with the MTLINC attribute.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'00A7'	
value	INSTANCE_OF REQUIRED	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

NAME

NAMDR — Name Data

DESCRIPTION (Semantic)

Dictionary	QDDPRMD
Codepoint	X'0066'
Length	*
Class	CLASS
Sprcls	FIELD - A Discrete Unit of Data

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 un-quoted character strings. Enclosing quotes are required when embedded blanks are in the name. Examples of name literals are *NAME*, *PAYROLL*, and *MESSAGE PROFILE*.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	INSTANCE_OF MAXLEN REQUIRED	CHRSTRDR - Character String 255
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXTNAM* on page 348
	PKGRPLVRS on page 518
	PRCNAM on page 525
	SRVNAM on page 727
	TYPDEFNAM on page 873
	TYPFMLNM on page 880
	VRSNAM on page 900
semantic	*INHERITANCE* on page 380

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0027'	
value	INSTANCE_OF MAXLEN OPTIONAL	CHRSTRDR - Character String 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DCTINDEN* on page 254
	DEFINITION on page 262
semantic	*APPCMNI* on page 68
	CCSIDMGR on page 140
	CHRDR on page 145
	DEFLST on page 263
	DICTIONARY on page 272
	INHERITANCE on page 380
	NAMDR on page 444
	SYNCMNI on page 774
	TCPCMNI on page 840
sprcls	*DFTRDBCOL* on page 268
	EXTNAM on page 348
	MGRNAM on page 434
	NEWPASSWORD on page 450

Variable	Reference
	PASSWORD on page 491
	PKGOWNID on page 512
	PKGRPLVRS on page 518
	RSYNCTYP on page 647
	SRVCLSNM on page 717
	TYPDEFNAM on page 873
	TYPFMLNM on page 880
	USRID on page 888
	VRSNAM on page 900

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	' '
	REPEATABLE	
	MAXLEN	255
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DFTRDBCOL* on page 268
	PKGID on page 504
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGOWNID on page 512
	RDBCOLID on page 586
	RDBNAM on page 589

NAME

NBRROW — Number of Fetch or Insert Rows

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'213A'

Length *

Class CLASS

Sprcls BIN - Binary Integer Number

Number of Fetch or Insert Rows (NBRROW) specifies the number of rows to insert for multi-row inserts. NBRROW also specifies the number of rows of data being requested for multi-row fetches.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'213A'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	NOTE	For multi-row fetches, values of 1 ... N return up to N rows of data.
	NOTE	For multi-row inserts, values of 1 ... N insert up to N rows of data.
	MINVAL	0
	NOTE	For multi-row fetches, a value of zero indicates that no rows of data are to be returned.
	NOTE	For multi-row inserts, a value of zero is not valid.
	DFTVAL	1
	NOTE	Fetch or insert one row of data.
	OPTIONAL	

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
	EXCSQLSTT on page 336
semantic	*CNTQRY* on page 217
	EXCSQLSTT on page 336

NEWPASSWORD

NAME

NEWPASSWORD — New Password

DESCRIPTION (Semantic)

Dictionary	QDDBASD
Codepoint	11DE
Length	*
Class	CLASS
Sprcls	NAME - Name

NEWPASSWORD at the Target System specifies the new password associated with the defined end-user name.

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.

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 the *USRSECOVR* on page 893 term.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11DE'	
value	INSTANCE_OF MINLEN MAXLEN REQUIRED	CHRSTRDR - Character String 1 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SECCHK* on page 652
semantic	*SECCHK* on page 652
	USRSECOVR on page 893

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	4	
class	X'002A'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*CLASS* on page 148
	SPRCLS on page 678

NOTE

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0014'	
value	INSTANCE_OF REQUIRED	CHRSTRDR - Character String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*MANAGER* on page 417
mgrdepls	*AGENT* on page 56
semantic	*COLLECTION* on page 231
	OBJOVR on page 464
	QLFATT on page 554

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

Variable	Reference
semantic	*INHERITANCE* on page 380
	OBJOVR on page 464
sprcls	*BIN* on page 110

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 the term *USRSECOVR* on page 893 for more information about the new password security mechanism.

SEE ALSO

Variable	Reference
semantic	*SECMEC* on page 661
	USRIDNWPWD on page 889
title	*USRIDNWPWD* on page 889

NAME

OBJDSS — Object Data Stream Structure

OBJDSS

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.

All fields of the OBJDSS header must be specified in the order shown in Figure 3-50 through Figure 3-52 on page 456because they are not self-defining structures.

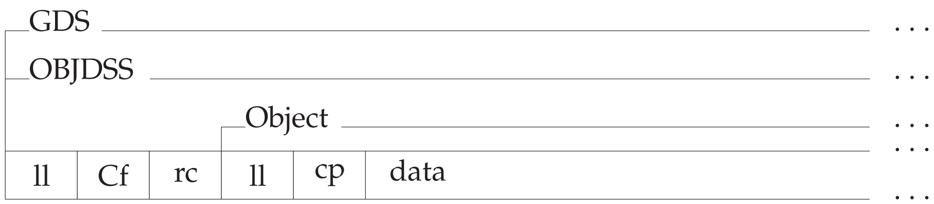


Figure 3-50 Data Objects

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.

GDS . . .										
OBJDSS . . .										
			RECAL							
					RECNBR			RECORD		
ll	Cf	rc	ll	cp	ll	cp	record number	ll	cp	record data

Figure 3-51 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 code point (RECAL, RECNBR, or RECORD).

data Object data value.

OBJDSS . . .					
			RECORD . . .		
cl	Cf	rc	ll	cp	record data . . .

. . . OBJDSS	
. . . RECORD	
nl	record data . . .

Figure 3-52 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 code point (of RECORD).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	6
	MAXVAL	32767
	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	DDMID - DDM Identifier
	REQUIRED	
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).
	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	BYTSTRDR - Byte String
	NOTE	The contents of this field are determined by the data object to be transmitted.
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PRCCNVCD* on page 521
semantic	*AGNCMDPR* on page 60
	APPCMNI on page 68
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	COMMAND on page 233
	DSS on page 289
	EXCSATRD on page 329
	FDOOBJ on page 360
	INHERITANCE on page 380
	LMTBLKPRC on page 400
	OBJNSPRM on page 462
	QRYBLK on page 555
	RDBOVR on page 593
	RQSCRR on page 626
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLSTT on page 714
	SYNCCRD on page 759
	SYNCLOG on page 764
	SYNCMNI on page 774
	SYNCRRD on page 826
	TCPCMNI on page 840
	TCPSRCCD on page 853
	TCPSRCCR on page 856
	TYPDEF on page 872

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 nsvar 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 the term SPACE for a description of how collections are encoded in spaces. And see the term DSS 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 (*nscmd*) 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)`

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	INSTANCE_OF LENGTH NOTE REQUIRED	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).
class	INSTANCE_OF NOTE REQUIRED	CODPNTDR - Code Point Data Representation Specifies the code point of the class of the object.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ARRAY* on page 96
	ASSOCIATION on page 98
	ATTLST on page 99
	QLFATT on page 554
semantic	*CLASS* on page 148
	DSS on page 289
	INHERITANCE on page 380
	NIL on page 451
	OBJOVR on page 464
	STRLYR on page 737
	SUBSETS on page 747
sprcls	*CMDTRG* on page 175
	COLLECTION on page 231
	IGNORABLE on page 378
	INHERITED on page 385

Variable	Reference
	MANAGER on page 417
	MTLEXC on page 442
	MTLINC on page 443
	OPTIONAL on page 485
	REPEATABLE on page 612
	REQUIRED on page 615
	SCALAR on page 649

OBJNSPRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1253'	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code 16 - SEVERE - Severe Error Severity Code
codpnt	INSTANCE_OF REQUIRED NOTE	CODPNT - Code Point This is the code point of the object that is not supported.
reccnt	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.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information

clscmd	**NIL**
inscmd	**NIL**

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
	BNDSQLSTT on page 130
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	REBIND on page 606
	SECCHK on page 652
semantic	*SUBSETS* on page 747

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 3-53 on page 465:

- 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 3-54 on page 466. 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:

OBJECT Self-Identifying Data (*OBJECT* on page 459)

SCALAR
Scalar Object (*SCALAR* on page 649)

COLLECTION
Collection Object (*COLLECTION* on page 231)

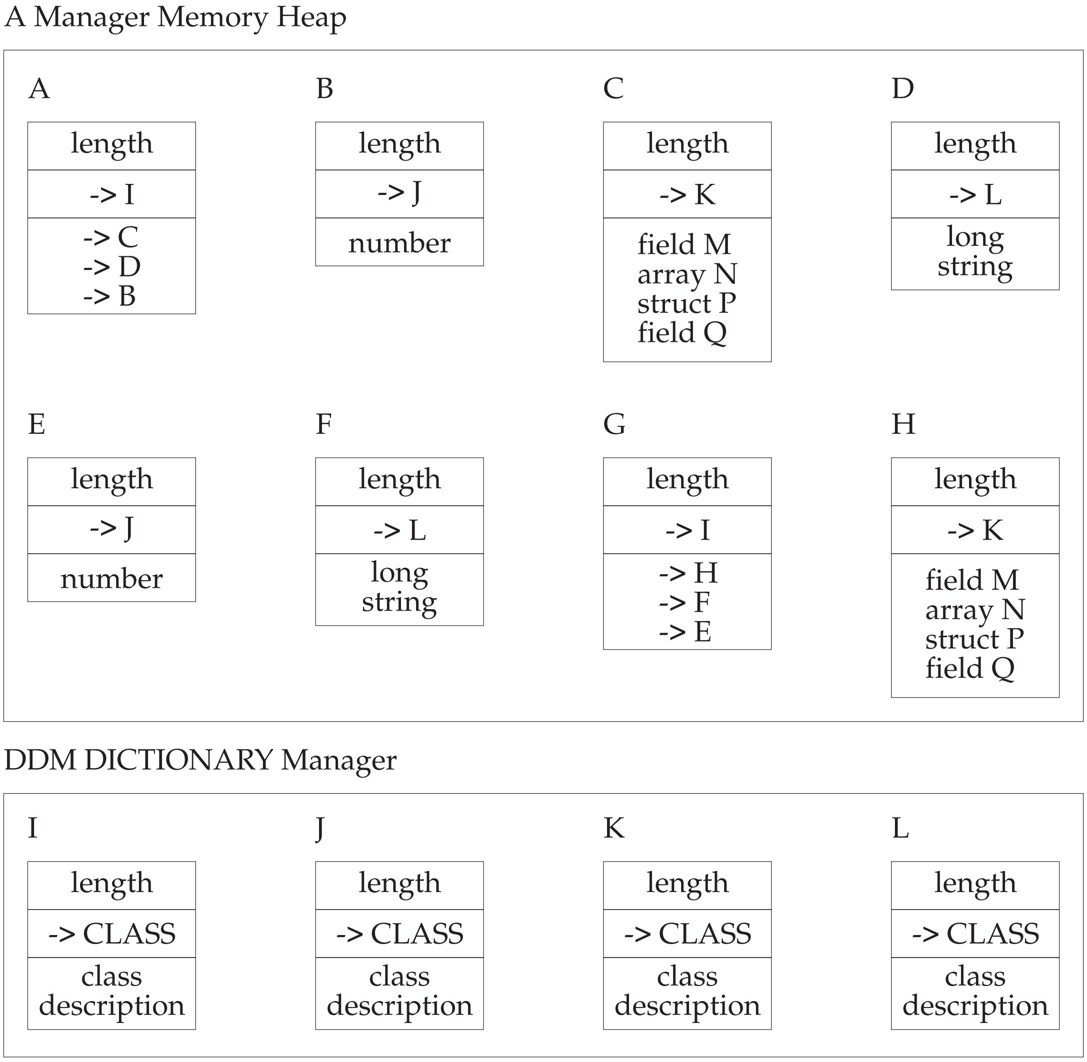


Figure 3-53 DDM Objects

Interchange Data Stream Form of Object A

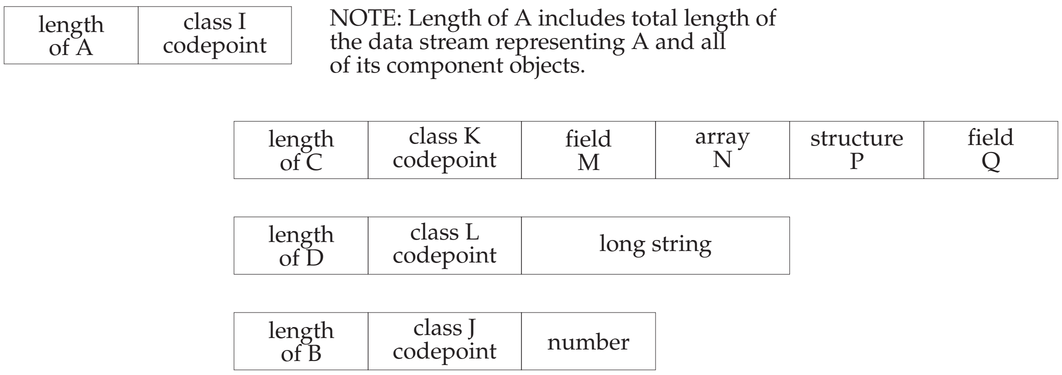


Figure 3-54 DDM Object Interchange Format

SEE ALSO

Variable	Reference
semantic	*DDM* on page 255

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 3-55 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.

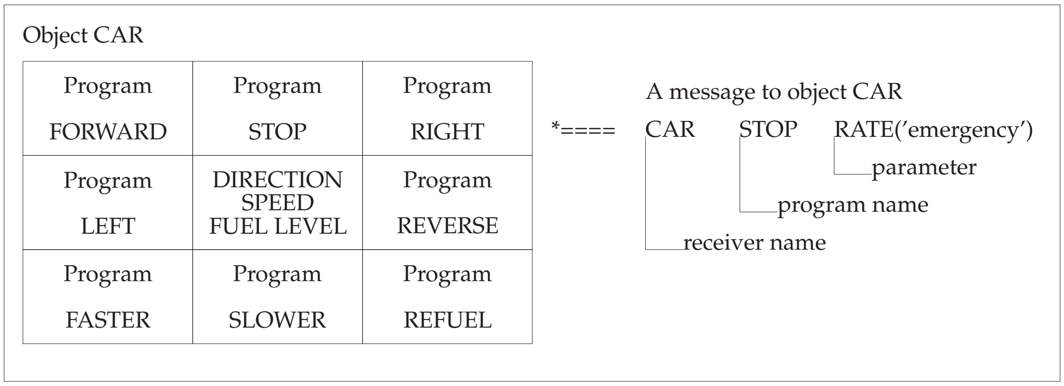


Figure 3-55 Objects—Data Items Encapsulated by Programs

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 3-56. 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.

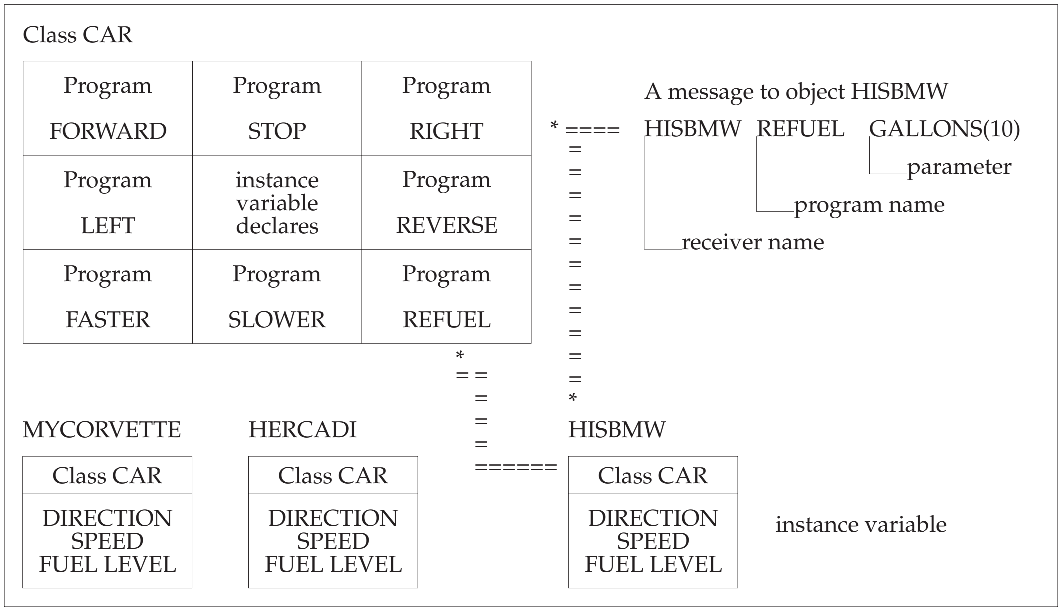


Figure 3-56 Objects as Instances of a Class

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 3-57 on page 469 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 3-58 on page 470. 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.

CLASSES	CLASS CHARACTERISTICS	INSTANCES	INSTANCE CHARACTERISTICS
Vehicle	mobile		
Water Vehicles	float		
Air Vehicles	fly		
Wheeled Vehicles	have wheels		
Truck	carries cargo		
Dump			
Milk			
etc.			
Passenger Vehicle	carries people		
Bus	up to 50 people		
CityBus	stops often		
		Bus 23	brown
		Bus 45	white
		Bus 99	dirty
HighwayBus			
etc.			
Car	carries up to 5 people		
Chevy	utility	Blue Bell	old
Corvette	sporty	Streak	quick
Cadillac	comfortable	Rose Bud	wire wheels
etc.			

Figure 3-57 Hierarchical Taxonomy

A hierarchical method of describing objects in the application domain brings organization to an apparent chaos of similarities and differences.

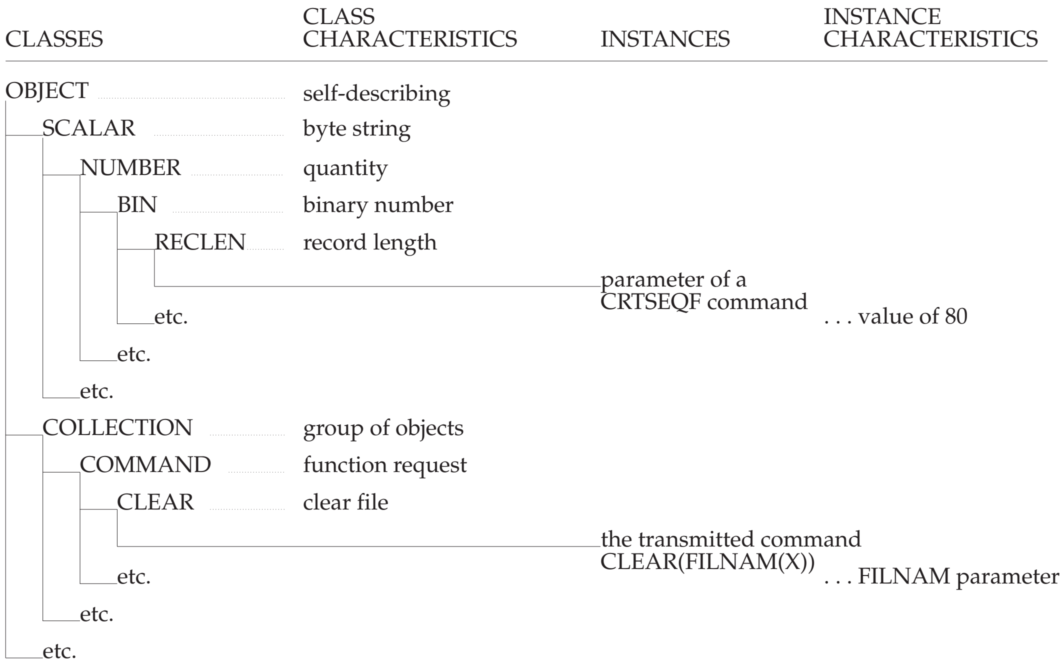


Figure 3-58 A Sample of the DDM Class Hierarchy

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 3-58.

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 3-59 on page 471 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.

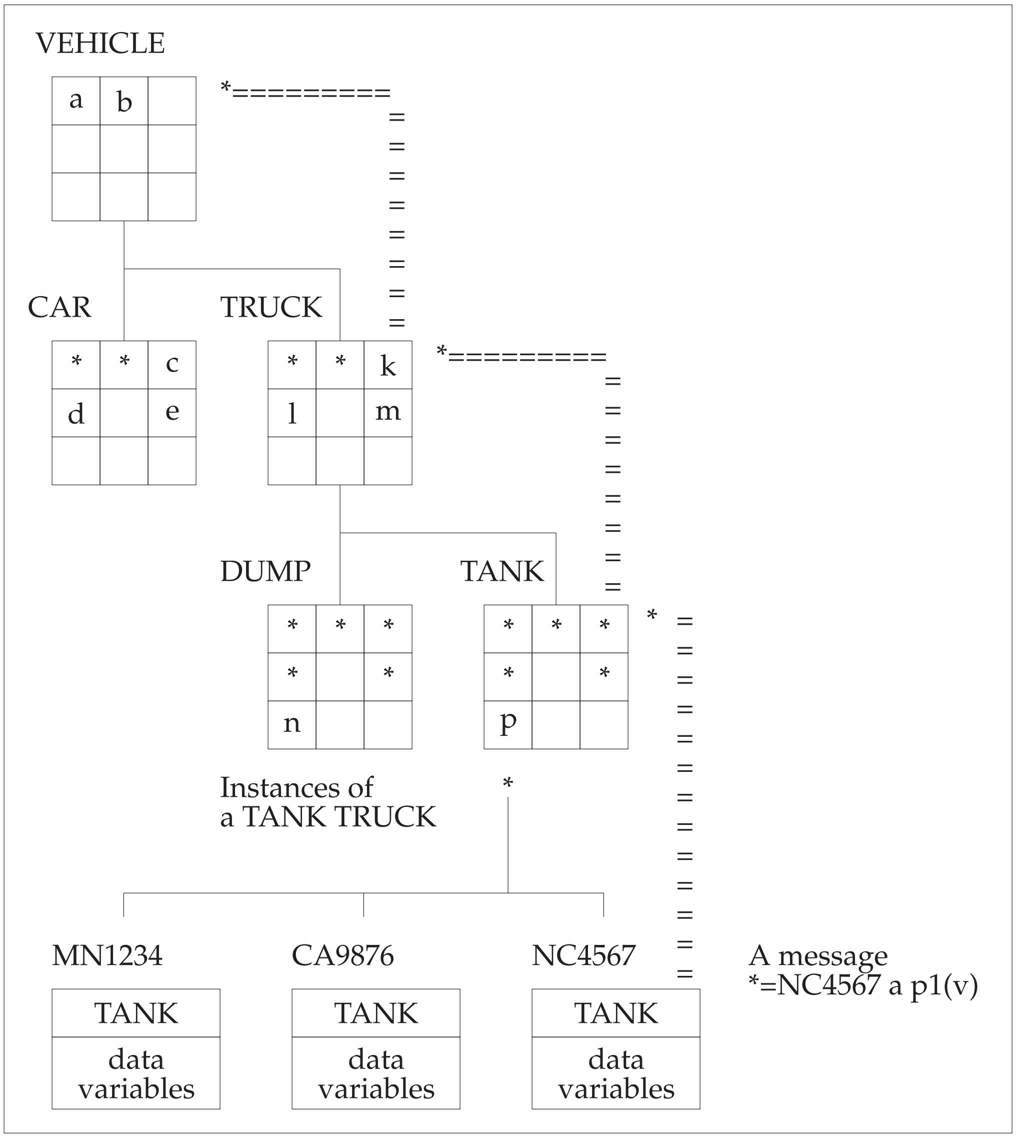


Figure 3-59 Inheritance of Programs from More Abstract Classes

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 3-60 on page 472.

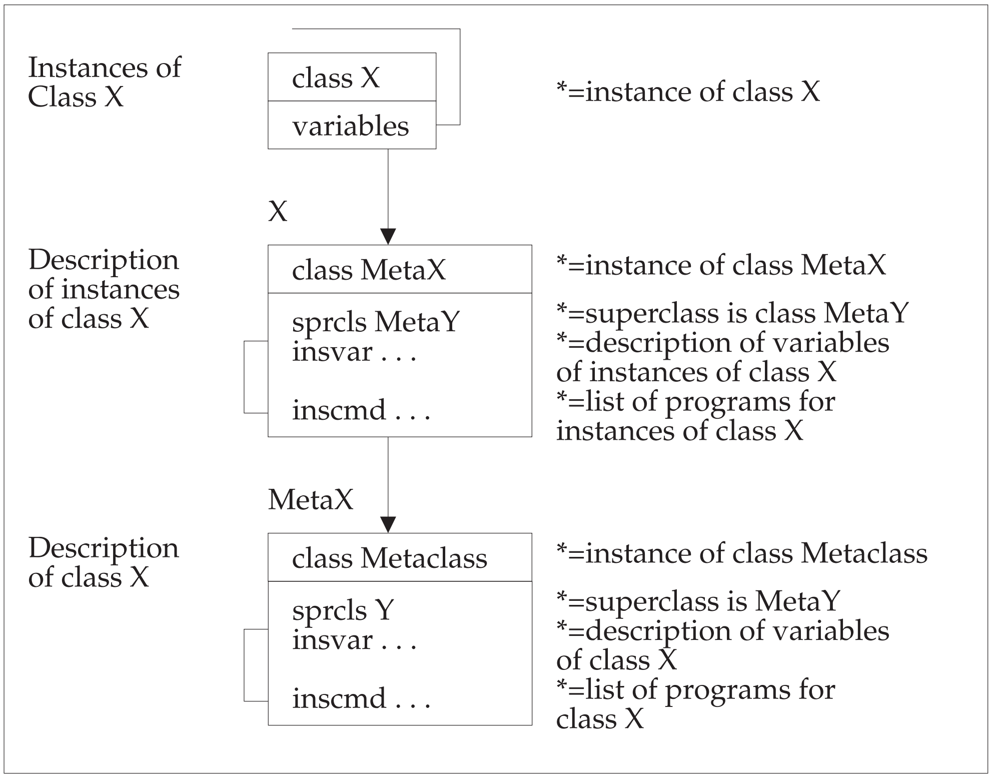


Figure 3-60 Smalltalk Instance, Class, and Metaclass Relationships

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 3-61 on page 473. Thus, the class describes the variables that can exist in a class and specifies the list of programs for the class.

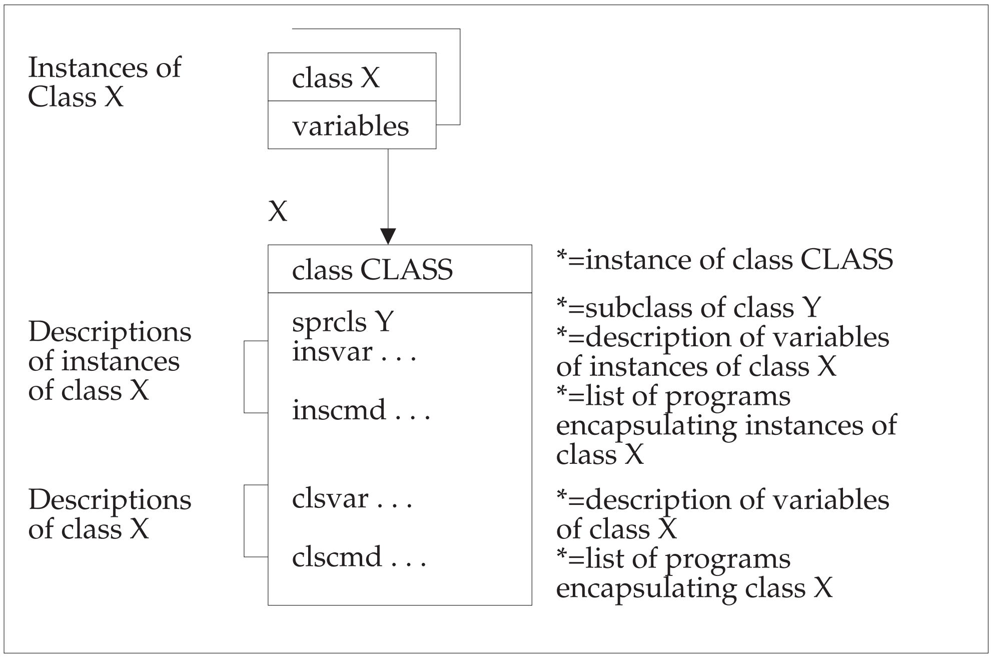


Figure 3-61 DDM Instance, Class, and Metaclass Relationships

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

None.

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2212'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*OPNQRY* on page 475
semantic	*FIXROWPRC* on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475

NAME

OPNQRY — Open Query

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'200C'
Length	*
Class	CLASS
Sprcls	COMMAND - Command

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 *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 *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 *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 the description of the term QRYBLK for a definition of a query block.

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 *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 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 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.

Normal completion of the OPNQRY command suspends the query and returns an OPNQRYRM followed by one or more QRYDSC and QRYDTA objects, depending on which query protocol the target selects. The OPNQRYRM indicates which query protocol the target SQLAM has selected to use for the query.

Table 3-12 on page 478 is a decision table that summarizes the OPNQRY'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 the SQLCARD or QRYDTA reply data objects.

If the last of the answer set data is being returned, 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.
- Space is available in the current query block for an ENDQRYRM and SQLCARD object.

Then an ENDQRYRM and an SQLCARD object are returned after any QRYDSC or QRYDTA reply data objects, and the query is terminated. If not enough space is available in the current query block for the reply message and SQLCARD object, then the reply message and SQLCARD object are saved, and the query is suspended.

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 PKGBPARM 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.

Summary Decision Table

The OPNQRY functions are summarized in the following decision table (Table 3-12 on page 478). 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, consisting of a set of conditions and a set of actions resulting from receiving the OPNQRY command. An *X* indicates the condition exists, and a *Y* indicates the resulting actions. For instance, case A has the single condition,*the query is in the not-opened state* and the resulting actions are *the OPNQRYRM is returned*, *QRYDSC is returned*, *the QRYDTA is next* and optionally, *the ENDQRYRM and SQLCARD are returned*, and *the query is in the suspended state*.

Table 3-12 OPNQRY Summary Decision Table (Part 1)

Conditions	Cases A	B	C	D	E	F
Query is in the not-opened state	X	X	X	X	X	
Query is in the suspended state						X
Package binding process is active		X				
No rows of data exist in the answer set[17]			X			
Error detected preventing the OPNQRY command from being initiated by the SQLAM				X		
RDB had an error preventing the query from opening prior to returning the OPNQRYRM					X	

Table 3-13 OPNQRY Summary Decision Table (Part 2)

Actions	Cases A	B	C	D	E	F
Query is in the suspended state	Y					Y
OPNQRYRM is returned	Y		Y			
One or more QRYDSC are returned	Y		Y			
QRYDTA is returned if space is available[19]	Y					
SQLCARD is returned	Y		Y		Y	
QRYPOPRM is returned						Y
Query is terminated, placed in the not-opened state		Y	Y	Y	Y	
PKGBPARM is returned		Y				
ENDQRYRM is returned			Y			
Reply message is returned, see mdrpy				Y		
OPNQFLRM is returned					Y	

Source System Reply Processing

Return any reply messages and data objects to the requester.

---

19. Only occurs for LMTBLKPRC.

19. If both the OPNQRYRM and ENDQRYRM are returned, another SQLCARD can optionally be returned following the OPNQRYRM.

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'200C'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
qryblksz	INSTANCE_OF REQUIRED	QRYBLKSZ - Query Block Size
qryblkctl	INSTANCE_OF ENUVAL DFTVAL NOTE OPTIONAL	QRYBLKCTL - Query Block Protocol Control X'2410' - FRCFIXROW - Force Fixed Row Query Protocol '' Means that the value specified in the package for this parameter is used to determine which query protocol is used for this query.
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks 0
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
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 ACCRDB command is used.

X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
	OPTIONAL	
X'2412'	INSTANCE_OF	SQLDTA - SQL Program Variable Data
	OPTIONAL	
	NOTE	Specified when the query has input variables.
rpydta		**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.
	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.
	OPTIONAL	
	REPEATABLE	
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.
	OPTIONAL	
	REPEATABLE	
X'2408'	INSTANCE_OF	SQLCARD - SQL Communications Area Reply Data
	OPTIONAL	
	NOTE	The SQLCARD object cannot be returned without also returning a reply message. The reply message returned with SQLCARD always precedes

		the SQLCARD object.
	REPEATABLE	
	NOTE	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 nonterminating error occurs, the QRYDTA object only contains an SQLCARD that reports the error.
cmdrpy		**COMMAND REPLIES**
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'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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*APPSRCCR* on page 82
	BGNBND on page 101
	CLSQRY on page 163
	CNTQRY on page 217
	ENDBND on page 307
	FIXROWPRC on page 365
	FRCFIXROW on page 370
	ISOLVLALL on page 390
	ISOLVLCHG on page 391
	ISOLVLCS on page 392
	ISOLVLNC on page 393
	ISOLVLRR on page 394
	LMTBLKPRC on page 400
	MAXBLKEXT on page 420
	OPNQFLRM on page 474
	OPNQRYRM on page 483
	QRYBLK on page 555
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	QRYDTA on page 561
	QRYPOPRM on page 564
	RDBCMM on page 582
	RDBRLLBCK on page 598
	SQL on page 680
	SQLAM on page 694

NAME

OPNQRYRM — Open Query Complete

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2205'

Length *

Class CLASS

Sprcls RPYMSG - Reply Message

Open Query Complete (OPNQRYRM) 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 OPNQRYRM is returned for each answer set.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2205'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
qryprctyp	INSTANCE_OF REQUIRED	QRYPRCTYP - Query Protocol Type
sqlcsrhld	INSTANCE_OF OPTIONAL	SQLCSRHLD - Hold Cursor Position
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSQLSTT* on page 336
	OPNQRY on page 475
rpydta	*OPNQRY* on page 475
semantic	*CNTQRY* on page 217

Variable	Reference
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYBLK on page 555

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'002D'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	COMMAND on page 233
	LVLCMP on page 412
	MGRLVLN on page 430

Variable	Reference
	ORDCOL on page 487
	SUBSETS on page 747

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.

clsvar	**NIL**
insvar	**CLASS INSTANCE VARIABLES**
length	4
class	X'004C'
clscmd	**NIL**
inscmd	**NIL**

SEE ALSO

Variable	Reference
sprcls	*ASSOCIATION* on page 98
	FDOOBJ on page 360
	TYPDEF on page 872

OSFDCE

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

Variable	Reference
semantic	*DCESEC* on page 247
	SECMEC on page 661
	SECMGR on page 663

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 rely 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.

clsvar	**NIL**	
insvar		**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
	ENUVAL	object.
	NOTE	X'F0' - FALSE - False State
		The requester does not expect the target
		SQLAM to return output within an SQLDTARD
	DFTVAL	reply data object.
	OPTIONAL	X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336

OWNER

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

Variable	Reference
insvar	*PKGATHRUL* on page 494
semantic	*PKGATHRUL* on page 494

NAME

PASSWORD — Password

DESCRIPTION (Semantic)

Dictionary	QDDBASD
Codepoint	11A1
Length	*
Class	CLASS
Sprcls	NAME - Name

Password at the Target System (PASSWORD) specifies the password associated with the defined end-user name.

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 the *USRSECOVR* on page 893 term.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11A1'	
value	INSTANCE_OF MINLEN MAXLEN REQUIRED	CHRSTRDR - Character String 1 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SECCHK* on page 652
semantic	*APPCMNI* on page 68
	SECCHK on page 652
	SYNCMNI on page 774
	USRSECOVR on page 893

PKGATHKP

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

Variable	Reference
insvar	*PKGATHOPT* on page 493
semantic	*PKGOWNID* on page 512
	PKGRPLALW on page 514

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'211E'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'2425' - PKGATHKP - Package Authorizations Keep
	ENUVAL	X'2424' - PKGATHRVK - Package Authorizations Revoked
	DFTVAL	X'2425' - PKGATHKP - Package Authorizations Keep
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	ENDBND on page 307
	PKGOWNID on page 512
	PKGRPLALW on page 514
	REBIND on page 606

PKGATHRUL

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 either the requester (the person executing the package) or the owner (the person who owns the package).

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 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'213F'	
value	INSTANCE_OF	BINDR - Binary Number Field
	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.
	DFTVAL	0 - REQUESTER - Package Requester Authorization Identifier
	OPTIONAL	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	REBIND on page 606

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

Variable	Reference
insvar	*PKGATHOPT* on page 493
semantic	*ENDBND* on page 307
	PKGOWNID on page 512
	PKGRPLALW on page 514

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2206'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*BNDSQLSTT* on page 130
	ENDBND on page 307
semantic	*BNDSQLSTT* on page 130
	ENDBND on page 307

NAME

PKGBPARM — RDB Package Binding Process Active

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2209'

Length *

Class CLASS

Sprcls RPYMSG - Reply Message

RDB Package Binding Process Active (PKGBPARM) 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 descriptions of terms BGNBND and ENDBND) before the command can be issued.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2209'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	EXCSQLIMM on page 331

Variable	Reference
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	REBIND on page 606
semantic	*BGNBND* on page 101
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	REBIND on page 606

PKGCNSTKN

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
name	INSTANCE_OF LENGTH	BYTSTRDR - Byte String 8
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGNAMCSN* on page 508
	PKGNAMCT on page 510
semantic	*PKGNAM* on page 507
	REBIND on page 606
	SQL on page 680

NAME

PKGDFTCC — 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 (PKGDFTCC) 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'119A'	
ccsidsbc	INSTANCE_OF OPTIONAL DFTVAL NOTE	CCSIDSBC - CCSID for Single-Byte Characters '' The default single-byte CCSID the target system defines is used.
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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2125'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'2432' - CSTSYSDFT - 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' - CSTSYSDFT - Character Subtype System Default
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDBRM* on page 48
	BGNBND on page 101
semantic	*ACCRDBRM* on page 48
	BGNBND on page 101
	ENDBND on page 307
	REBIND on page 606

PKGID

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
name	INSTANCE_OF	NAMSYMDR - Name Symbol Data Representation
	LENGTH	18
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGNAM* on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
semantic	*SQL* on page 680

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.

```
ISOLVLRR     <-- 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X‘2124’	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X‘2444’ - ISOLVLRR - 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
clscmd	**NIL**	

inscmd	**NIL**

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	REBIND on page 606

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	58	
class	X'210A'	
rdbnam	INSTANCE_OF LENGTH NOTE	NAMSYMDR - Name Symbol Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.
rdbcolid	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
pkgid	INSTANCE_OF	PKGID - RDB Package Identifier
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DRPPKG* on page 274
	REBIND on page 606
semantic	*DRPPKG* on page 274
	REBIND on page 606
	SQL on page 680

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	68	
class	X'2113'	
rdbnam	INSTANCE_OF LENGTH NOTE	NAMSYMDR - Name Symbol Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.
rdbcolid	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
pkgid	INSTANCE_OF	PKGID - RDB Package Identifier
pkgcnstkn	INSTANCE_OF	PKGCNSTKN - RDB Package Consistency Token
pkgsn	INSTANCE_OF	PKGSN - RDB Package Section Number
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BNDSQLSTT* on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DSCSQLSTT on page 285

Variable	Reference
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PKGSNLST on page 520
	PRCNAM on page 525
	PRPSQLSTT on page 533
	QRYNOPRM on page 562
	QRYPOPRM on page 564
semantic	*BNDSQLSTT* on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	LMTBLKPRC on page 400
	OPNQRY on page 475
	PRCNAM on page 525
	PRPSQLSTT on page 533
	SQL on page 680

PKGNAMCT

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	66	
class	X'2112'	
rdbnam	INSTANCE_OF LENGTH NOTE	NAMSYMDR - Name Symbol Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.
rdbcolid	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
pkgid	INSTANCE_OF	PKGID - RDB Package Identifier
pkgcnstkn	INSTANCE_OF	PKGCNSTKN - RDB Package Consistency Token
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	BGNBNDRM on page 109
	ENDBND on page 307
semantic	*BGNBND* on page 101
	BNDSQLSTT on page 130
	ENDBND on page 307
	PKGATHOPT on page 493

Variable	Reference
	PKGRPLALW on page 514
	PKGRPLOPT on page 516
	SQL on page 680

PKGOWNID

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 and ENDBND).
- If the rebind process results in the package being replaced.

See the terms PKGATHKP and PKGATHRVK (values of the PKGATHOPT parameter) for a description of existing package authorizations.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2131'	
value	INSTANCE_OF	NAMSYMDR - Name Symbol Data Representation
	MINLEN	1
	MAXLEN	8
	DFTVAL	''
	NOTE	Means that the default is the end-user name (identifier) of the requester that initiated the bind process.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	ENDBND on page 307
	PKGATHKP on page 492
	PKGATHRVK on page 496
	REBIND on page 606
	SQL on page 680

PKGRPLALW

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 and PKGATHRVK (values of the PKGATHOPT parameter) for explanations on how that affects existing package authorizations.

SEE ALSO

Variable	Reference
insvar	*PKGRPLOPT* on page 516
semantic	*BNDCHKONL* on page 120
	ENDBND on page 307
	PKGATHOPT on page 493
	PKGOWNID on page 512
	REBIND on page 606

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

Variable	Reference
insvar	*BGNBND* on page 101
	PKGRPLOPT on page 516
	PKGRPLVRS on page 518
semantic	*BGNBND* on page 101

PKGRPLOPT

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'211C'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'241F' - PKGRPLALW - Package Replacement Allowed
	ENUVAL	X'2420' - PKGRPLNA - Package Replacement Not Allowed
	DFTVAL	X'241F' - PKGRPLALW - Package Replacement Allowed
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	BNDCHKONL on page 120
	ENDBND on page 307
	PKGATHOPT on page 493
	PKGOWNID on page 512

Variable	Reference
	REBIND on page 606

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'212D'	
value	INSTANCE_OF MAXLEN DFTVAL NOTE NOTE	NAMDR - Name Data 254 '' If a value is not specified for this parameter, the VRSNAM parameter on this command assumes its default value. This parameter is ignored when PKGRPLOT(PKGRPLNA) is specified.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	PKGRPLALW on page 514
	PKGRPLOPT on page 516

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
sctnbr	INSTANCE_OF LENGTH MINVAL	BINDR - Binary Number Field 16 1
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGNAMCSN* on page 508

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2139'	
pkgnamcsn	INSTANCE_OF REPEATABLE	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RSLSETRM* on page 642
semantic	*LMTBLKPRC* on page 400

NAME

PRCCNVCD — Conversational Protocol Error Code

DESCRIPTION (Semantic)

Dictionary	QDDBASD
Codepoint	X'113F'
Length	*
Class	CLASS
Sprcls	STRING - String

Conversational 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'
	NOTE	RPYDSS received by target communications manager.
	ENUVAL	X'02'
	NOTE	Multiple DSSs sent without chaining or multiple DSS chains sent.
	ENUVAL	X'03'
	NOTE	OBJDSS sent when not allowed.
	ENUVAL	X'04'
	NOTE	Request correlation identifier of an RQSDSS is less than or equal to the previous RQSDSS's request correlation identifier in the chain.
	NOTE	If two RQSDSSs have the same request correlation identifier, the PRCCNVRM must be sent in a RPYDSS with a request correlation identifier of minus one.
	ENUVAL	X'05'
	NOTE	Request correlation identifier of an OBJDSS does not equal the request correlation identifier of the preceding RQSDSS.
	ENUVAL	X'06'
	NOTE	EXCSAT was not the first command after the connection was established.
	ENUVAL	X'10'
	MINLVL	5
	NOTE	ACCSEC or SECCHK command sent in wrong state. (See SECMGR for rules).
	ENUVAL	X'11'
	MINLVL	5

	NOTE	SYNCCTL or SYNCRSY command is used incorrectly. (See SYNCPTOV for rules).
	REQUIRED	
	ENUVAL	X'12'
	MINLVL	5
	NOTE	RDBNAM mismatch between ACCSEC, SECCHK, and ACCRDB.
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PRCCNVRM* on page 523
	SYNERRCD on page 831

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1245'	
svrcod	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
prccnvcd	INSTANCE_OF REQUIRED	PRCCNVCD - Conversational Protocol Error Code
reccnt	INSTANCE_OF MINVAL OPTIONAL NOTE MINLVL	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. 3
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
insvar	*PRCCNVCD* on page 521
	SYNERRCD on page 831
semantic	*DSS* on page 289
	EXCSAT on page 323
	PRCCNVCD on page 521
	SECMGR on page 663

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2138'	
value	INSTANCE_OF	NAMDR - Name Data
	OPTIONAL	
	NOTE	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 *pkgnamcsn* parameter on the EXCSQLSTT command.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336

PRCOVR

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 3-62 illustrates how an application program accesses a file as if all access method services and data management services are provided locally.

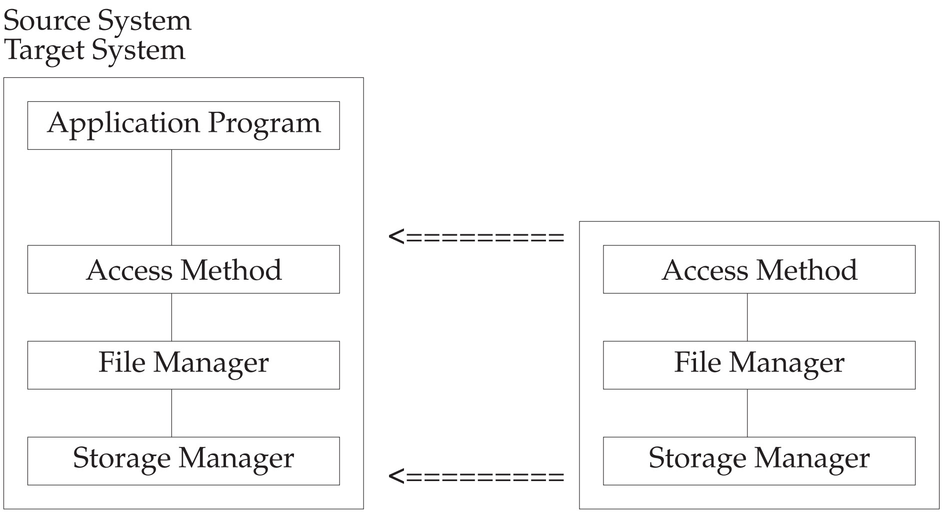


Figure 3-62 Transparent Data Management Services

To accomplish this, the DDM architecture is designed in terms of abstract services that the local system appears to perform. Figure 3-63 on page 527 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.

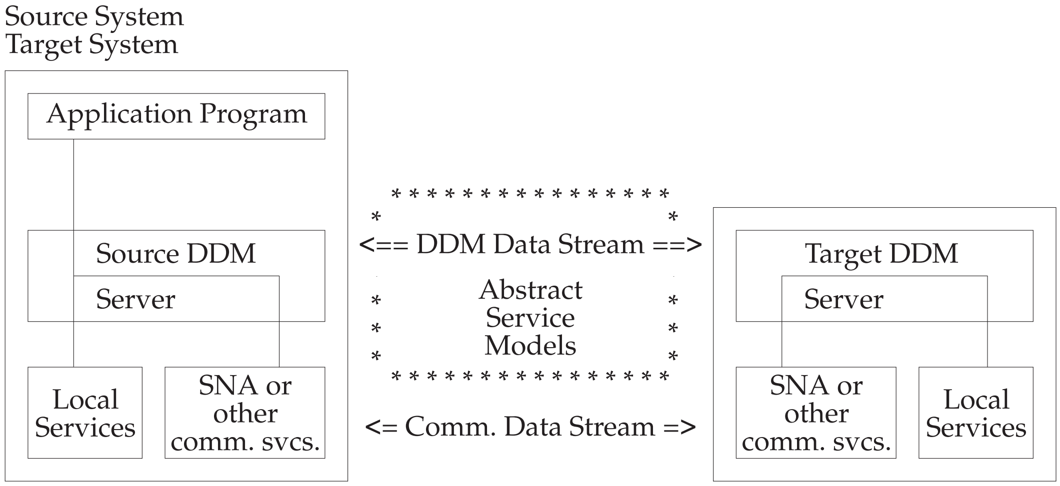


Figure 3-63 DDM Processing Overview

Figure 3-63 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 the term SUBSETS for a discussion of the subsetting rules of DDM architecture.

SEE ALSO

Variable	Reference
semantic	*CONCEPTS* on page 237

PRDDTA

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2104'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
semantic	*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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'112E'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 8
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
	ACCRDBRM on page 48
	RSCLMTRM on page 634
semantic	*ACCRDB* on page 42
	ACCRDBRM on page 48

PRMDMG

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

Variable	Reference
insvar	*AGNPRMRM* on page 61
	CMDCHKRM on page 170
	RSCLMTRM on page 634
	SVRCOD on page 756

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 the description of *SUBSETS* on page 747).

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1251'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
codpnt	INSTANCE_OF REQUIRED NOTE	CODPNT - Code Point Specifies the code point of the parameter not supported.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101

Variable	Reference
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
semantic	*SUBSETS* on page 747

NAME

PRPSQLSTT — Prepare SQL Statement

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'200D'
Length	*
Class	CLASS
Sprcls	COMMAND - Command

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 *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 *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 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.

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 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

Exception conditions the RDB detects are reported in the SQLDARD reply data object.

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. then

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'200D'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
rtnsqlda	INSTANCE_OF OPTIONAL	RTNSQLDA - Return SQL Descriptor Area

clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
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'2414'	INSTANCE_OF REQUIRED	SQLSTT - SQL Statement
rpydta		**REPLY OBJECTS**
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'2411'	INSTANCE_OF OPTIONAL MTLEXC NOTE	SQLDARD - SQLDA Reply Data X'2408' - SQLCARD - SQL Communications Area Reply Data Returned if RTNSQLDA(TRUE) is specified.
X'2408'	INSTANCE_OF OPTIONAL	SQLCARD - SQL Communications Area Reply Data

	MTLEXC NOTE	X'2411' - SQLDARD - SQLDA Reply Data Returned if RTNSQLDA(FALSE) is specified or if ABNUOWRM is also returned.
cmdrpy		**COMMAND REPLIES**
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	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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	SQL on page 680
	SQLAM on page 694

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 the term *USRSECOVR* on page 893 for more information about the password security mechanism.

SEE ALSO

Variable	Reference
semantic	*SECMEC* on page 661
	USRIDNWPWD on page 889
	USRIDPWD on page 891
title	*USRIDNWPWD* on page 889
	USRIDPWD on page 891

QDDBASD

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.

mgrlvln	1	
mgrdepls	INSTANCE_OF	QDDPRMD - DDM Primitive Classes Dictionary
	MGRLVLN	1
	NOTE	The primitives dictionary includes class descriptions for many classes on which QDDBASD classes depend.
objlst	*	
dctind	*	

SEE ALSO

Variable	Reference
None.	*ACCSEC* on page 51
	ACCSECRD on page 55
	AGENT on page 56
	AGNPRMRM on page 61
	CCSIDDBC on page 138
	CCSIDMBC on page 139
	CCSIDMGR on page 140
	CCSIDSBC on page 144
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	CMDNSPRM on page 173
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	DCTIND on page 253
	DCTINDEN on page 254

Variable	Reference
	DEPERRCD on page 265
	DICTIONARY on page 272
	DSSFMT on page 301
	EXCSAT on page 323
	EXCSATRD on page 329
	EXTNAM on page 348
	FDODTA on page 357
	FDOOBJ on page 360
	IPADDR on page 388
	LOGNAME on page 410
	LOGTSTMP on page 411
	MANAGER on page 417
	MGRDEPRM on page 426
	MGRLVL on page 427
	MGRLVLLS on page 429
	MGRLVLN on page 430
	MGRLVLRM on page 432
	MGRNAM on page 434
	NEWPASSWORD on page 450
	OBJDSS on page 455
	OBJNSPRM on page 462
	PASSWORD on page 491
	PKGDFTCC on page 501
	PKGRPLVRS on page 518
	PRCCNVCD on page 521
	PRCCNVRM on page 523
	PRCNAM on page 525
	PRDID on page 529
	PRMNSPRM on page 531
	RDBCMTOK on page 585
	RLSCONV on page 619
	RPYDSS on page 620
	RPYMSG on page 624

Variable	Reference
	RQSCRR on page 626
	RQSDSS on page 629
	RSCLMTRM on page 634
	RSCNAM on page 637
	RSCTYP on page 638
	RSNCOD on page 643
	RSYNCTYP on page 647
	SECCHK on page 652
	SECCHKCD on page 656
	SECCHKRM on page 659
	SECMEC on page 661
	SECMGR on page 663
	SECMGRNM on page 666
	SECTKN on page 669
	SERVER on page 670
	SNAADDR on page 674
	SPVNAM on page 679
	SRVCLSNM on page 717
	SRVDGN on page 720
	SRVNAM on page 727
	SRVRLSLV on page 731
	STGLMT on page 732
	SUPERVISOR on page 753
	SVCERRNO on page 755
	SVRCOD on page 756
	SYNCCRD on page 759
	SYNCCTL on page 760
	SYNCLOG on page 764
	SYNCPTMGR on page 782
	SYNCRRD on page 826
	SYNCRSY on page 828
	SYNCTYPE on page 830
	SYNERRCD on page 831
	SYNTAXRM on page 835
	TCPHOST on page 846

Variable	Reference
	TEXT on page 864
	TRGNSPRM on page 868
	UOWID on page 883
	UOWSTATE on page 885
	USRID on page 888
	VALNSPRM on page 898
	VRSNAM on page 900
semantic	*CODPNTDR* on page 228
	EXCSAT on page 323
	SUBSETS on page 747

QDDPRMD

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.

mgrlvln	1
mgrdepls	**NIL**
objlst	*
dctind	*

SEE ALSO

Variable	Reference
None.	*ACCDMG* on page 41
	ARRAY on page 96
	ASSOCIATION on page 98
	ATTLST on page 99
	BIN on page 110
	BINDR on page 112
	BITDR on page 115
	BITSTRDR on page 117
	BOOLEAN on page 136
	BYTDR on page 137
	CHRDR on page 145
	CHRSTRDR on page 146
	CLASS on page 148
	CMDTRG on page 175
	CNSVAL on page 216
	CODPNT on page 225
	CODPNTDR on page 228
	COLLECTION on page 231
	COMMAND on page 233
	CONSTANT on page 238
	DATA on page 245

Variable	Reference
	DCESEC on page 247
	DDMID on page 258
	DEFINITION on page 262
	DEFLST on page 263
	DFTVAL on page 269
	ELMCLS on page 306
	ENUCLS on page 316
	ENULEN on page 317
	ENUVAL on page 318
	ERROR on page 319
	FALSE on page 349
	FDODSC on page 354
	FIELD on page 363
	HELP on page 371
	HEXDR on page 374
	HEXSTRDR on page 376
	IGNORABLE on page 378
	INFO on page 379
	INHERITED on page 385
	INSTANCE_OF on page 387
	LENGTH on page 398
	MAXLEN on page 422
	MAXVAL on page 425
	MINLEN on page 439
	MINLVL on page 440
	MINVAL on page 441
	MTLEXC on page 442
	MTLINC on page 443
	NAMDR on page 444
	NAME on page 445
	NAMSYMDR on page 447
	NIL on page 451
	NOTE on page 452
	NUMBER on page 453

Variable	Reference
	OBJECT on page 459
	OPTIONAL on page 485
	ORDCOL on page 487
	OWNER on page 490
	PRMDMG on page 530
	QLFATT on page 554
	REPEATABLE on page 612
	REQUESTER on page 614
	REQUIRED on page 615
	RESERVED on page 617
	SCALAR on page 649
	SESDMG on page 672
	SEVERE on page 673
	SPCVAL on page 676
	SPRCLS on page 678
	STRING on page 735
	TITLE on page 865
	TRUE on page 870
	TYPDEF on page 872
	TYPDEFNAM on page 873
	TYPDEFOVR on page 876
	TYPFMLNM on page 880
	USRIDNWPWD on page 889
	USRIDONL on page 890
	USRIDPWD on page 891
	WARNING on page 902
semantic	*CODPNTDR* on page 228
	EXCSAT on page 323
	SUBSETS on page 747

NAME

QDDRDBD — DDM Relational Database Classes Dictionary

DESCRIPTION (Semantic)

Dictionary QDDSRVDR

Length *

Class DICTIONARY

This dictionary contains all DDM classes that describe the commands, replies, and data objects required for relational database functions in DDM.

mgrlvln	3	
mgrdepls	INSTANCE_OF MGRLVLN NOTE	QDDPRMD - DDM Primitive Classes Dictionary 1 The primitives dictionary includes class descriptions for many classes on which QDDRDBD classes depend.
	INSTANCE_OF MGRLVLN NOTE	QDDBASD - DDM Base Classes Dictionary 1 The primitives dictionary includes class descriptions for many classes on which QDDRDBD classes depend.
objlst	*	
dctind	*	

SEE ALSO

Variable	Reference
None.	*ABNUOWRM* on page 39
	ACCRDB on page 42
	ACCRDBRM on page 48
	BGNBND on page 101
	BGNBNDRM on page 109
	BNDCHKEXS on page 119
	BNDCHKONL on page 120
	BNDCRTCTL on page 121
	BNDERRALW on page 122
	BNDEXPOPT on page 123
	BNDEXSOPT on page 124
	BNDEXSRQR on page 125
	BNDNERALW on page 126

Variable	Reference
	BNDOPT on page 127
	BNDOPTNM on page 128
	BNDOPTVL on page 129
	BNDSQLSTT on page 130
	BNDSTTASM on page 135
	CLSQRY on page 163
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	CMMTYP on page 178
	CNTQRY on page 217
	CRRTKN on page 240
	CSTBITS on page 241
	CSTMBCS on page 242
	CSTSBCS on page 243
	CSTSYSDFT on page 244
	DECDELCMA on page 259
	DECDELPRD on page 260
	DECPRC on page 261
	DFTPKG on page 267
	DFTRDBCOL on page 268
	DGRIOPRL on page 270
	DRPPKG on page 274
	DSCERRCD on page 277
	DSCINVRM on page 279
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DTAMCHRM on page 303
	ENDBND on page 307
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	EURDATFMT on page 321
	EURTIMFMT on page 322
	EXCSQLIMM on page 331

Variable	Reference
	EXCSQLSTT on page 336
	EXPALL on page 344
	EXPNON on page 345
	FDODSCOFF on page 356
	FDODTAOFF on page 359
	FDOPRMOFF on page 361
	FDOTRPOFF on page 362
	FIXROWPRC on page 365
	FRCFIXROW on page 370
	ISODATFMT on page 389
	ISOLVLALL on page 390
	ISOLVLCHG on page 391
	ISOLVLCS on page 392
	ISOLVLNC on page 393
	ISOLVLRR on page 394
	ISOTIMFMT on page 395
	JISDATFMT on page 396
	JISTIMFMT on page 397
	LMTBLKPRC on page 400
	MAXBLKEXT on page 420
	MAXRSLCNT on page 423
	MAXSCTNBR on page 424
	NBRROW on page 449
	OPNQFLRM on page 474
	OPNQRY on page 475
	OPNQRYRM on page 483
	OUTEXP on page 489
	PKGATHKP on page 492
	PKGATHOPT on page 493
	PKGATHRUL on page 494
	PKGATHRVK on page 496
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGCNSTKN on page 500

Variable	Reference
	PKGDFTCST on page 503
	PKGID on page 504
	PKGISOLVL on page 505
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGOWNID on page 512
	PKGRPLALW on page 514
	PKGRPLNA on page 515
	PKGRPLOPT on page 516
	PKGSN on page 519
	PKGSNLST on page 520
	PRDDTA on page 528
	PRPSQLSTT on page 533
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	QRYDSC on page 560
	QRYDTA on page 561
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	QRYPRCTYP on page 566
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDB on page 571
	RDBACCCL on page 577
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBALWUPD on page 580
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBCOLID on page 586
	RDBNACRM on page 587
	RDBNAM on page 589
	RDBNFNRM on page 592

Variable	Reference
	RDBRLLBCK on page 598
	RDBRLSCMM on page 601
	RDBRLSCNV on page 602
	RDBRLSOPT on page 603
	RDBUPDRM on page 604
	REBIND on page 606
	RSLSETFLG on page 639
	ISOLVLRR on page 394
	ISOTIMFMT on page 395
	JISDATFMT on page 396
	JISTIMFMT on page 397
	LMTBLKPRC on page 400
	MAXBLKEXT on page 420
	MAXRSLCNT on page 423
	MAXSCTNBR on page 424
	NBRROW on page 449
	OPNQFLRM on page 474
	OPNQRY on page 475
	OPNQRYRM on page 483
	OUTEXP on page 489
	PKGATHKP on page 492
	PKGATHOPT on page 493
	PKGATHRUL on page 494
	PKGATHRVK on page 496
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGCNSTKN on page 500
	PKGDFTCST on page 503
	PKGID on page 504
	PKGISOLVL on page 505
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGOWNID on page 512
	PKGRPLALW on page 514

Variable	Reference
	PKGRPLNA on page 515
	PKGRPLOPT on page 516
	PKGSN on page 519
	PKGSNLST on page 520
	PRDDTA on page 528
	PRPSQLSTT on page 533
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	QRYDSC on page 560
	QRYDTA on page 561
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	QRYPRCTYP on page 566
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDB on page 571
	RDBACCCL on page 577
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBALWUPD on page 580
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBCOLID on page 586
	RDBNACRM on page 587
	RDBNAM on page 589
	RDBNFNRM on page 592
	RDBRLLBCK on page 598
	RDBRLSCMM on page 601
	RDBRLSCNV on page 602
	RDBRLSOPT on page 603
	RDBUPDRM on page 604
	REBIND on page 606
	RSLSETFLG on page 639
	RSLSETRM on page 642

Variable	Reference
	RTNSQLDA on page 648
	SQL on page 680
	SQLAM on page 694
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLCSRHLD on page 706
	SQLDARD on page 707
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLERRRM on page 710
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714
	SQLSTTNBR on page 715
	SQLSTTVRB on page 716
	SRVLCNT on page 722
	SRVLSRV on page 723
	SRVLST on page 725
	SRVPRTY on page 730
	STRDELAP on page 733
	STRDELDQ on page 734
	STTASMEUI on page 740
	STTDATFMT on page 741
	STTDECDEL on page 742
	STTSCCCLS on page 743
	STTSTRDEL on page 744
	STTTIMFMT on page 746
	TRGDFTRT on page 867
	UOWDSP on page 882
	USADATFMT on page 886
	USATIMFMT on page 887
semantic	*CODPNTDR* on page 228
	EXCSAT on page 323
	SUBSETS on page 747

QDDTTRD

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

Variable	Reference
None.	*ABBREVIATIONS* on page 30
	AGNCMDPR on page 60
	AGNRPYPR on page 63
	APPCMNFL on page 64
	APPCMNI on page 68
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CMNLYR on page 189
	CMNOVR on page 196
	CONCEPTS on page 237
	DCESECOVR on page 248
	DCESECTKN on page 252
	DDM on page 255
	DSS on page 289
	DTAOVR on page 305
	EXTENSIONS on page 346
	FDOCA on page 351
	INHERITANCE on page 380
	LVLCMP on page 412
	MGROVR on page 436

Variable	Reference
	NWPWDSEC on page 454
	OBJOVR on page 464
	OSFDCE on page 488
	PRCOVR on page 526
	PWDSEC on page 537
	QRYBLK on page 555
	RDBOVR on page 593
	RESYNOVR on page 618
	SECOVR on page 667
	SNASECOVR on page 675
	SRVOVR on page 728
	STRLYR on page 737
	SUBSETS on page 747
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCMNI on page 774
	SYNCMNT on page 778
	SYNCPTOV on page 787
	TCPCMNFL on page 838
	TCPCMNI on page 840
	TCPCMNT on page 844
	TCPIPOVR on page 847
	TCPSRCCD on page 853
	TCPSRCCR on page 856
	TCPSRCER on page 859
	TCPTRGER on page 861
	USRIDSEC on page 892
	USRSECOVR on page 893

QLFATT

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 document, 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.
```

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘0007’	
key	SPRCLS REQUIRED	OBJECT - Self-Identifying Data
value	INSTANCE_OF REQUIRED	ATTLST - Attribute List
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

None.

NAME

QRYBLK — Query Block

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length *

Class HELP

Query Block (QRYBLK) is a logical construct mapped over the data stream structures which the communications manager handles. Figure 3-64 shows four example mappings of query blocks to the data stream structures that the communications manager receives.

Case 1:

```
RPYDSS( OPNQRYRM(-)) OBJDSS( QRYDSC(-))
|<------------------ one block ------------------>|
```

Case 2:

```
RPYDSS( OPNQRYRM(-)) OBJDSS(QRYDSC(---)) OBJDSS( QRYDSC(----))
|<------------------ one block ------------------->|<--- one block --->|
```

Case 3:

```
RPYDSS( ENDQRYRM(-)) OBJDSS(SQLCARD(-))
|<------------------ one block ------------------->|
```

Case 4:

```
OBJDSS( QRYDTA(-------------------------------------------------------------------))
|<------------------------------ one block ------------------------------>|
```

Figure 3-64 Query Block Examples

Query blocks have the following characteristics:

- The size of a query block is specified by the *qryblksz* parameter on the OPNQRY, CNTQRY, and EXCSQLSTT commands. The size of query blocks can be changed on subsequent CNTQRY commands.
- The *qryblksz* parameter specifies the size of all query blocks an OPNQRY or CNTQRY command returns except possibly the last query block. The last query block can be smaller than the specified size. No query block can be larger than the specified size. See the term LMTBLKPRC for the specific rules on using the space within the last QRYBLK. The *qryblksz* parameter specifies the size of all query blocks an EXCSQLSTT command returns except possibly the blocks of the summary component, a block preceding a block that contains an OPNQRYRM, or the last query block. These blocks can be smaller than the specified size. No query block can be larger than the specified size. See the term LMTBLKPRC for the specific rules on using the space within blocks that are smaller than the size specified by QRYBLKSZ.

- Every query block an OPNQRY, CNTQRY, or EXCSQLSTT command returns must start at the beginning of a data stream structure (DSS).
- Every query block an OPNQRY, CNTQRY, or EXCSQLSTT command returns must end at the end of a DSS.
- A query block contains one or more complete DSSs. See cases 1 and 3 in Figure 3-64 on page 555.
- A query block contains at most one QRYDTA object. See case 4 in Figure 3-64 on page 555.
- A query block contains at most one QRYDSC object. See case 2 in Figure 3-64 on page 555.
- An RPYDSS containing a reply message (except the OPNQRYRM, QRYPOPRM, or RSLSETRM) and the next OBJDSS containing the required SQLCARD object must be contained in a single final query block. If they cannot be contained in the final query block, then they are not returned until the next CNTQRY command is received.

The main reason for using the query block concept is to control the amount of data that is returned when LMTBLKPRC is being used. The query block size limits the amount of additional answer set data that can be returned after the last column of a row has been placed in a query block.

The source communications manager also uses the query block size to determine the amount of data to receive from the source communications facility. This allows the source communications manager to make the most efficient use of its receive buffers. However, the communications manager is not required to use the specified query block size.

See the following terms:

CNTQRY
Continue Query (*CNTQRY* on page 217)

OPNQRY
Open Query (*OPNQRY* on page 475)

LMTBLKPRC
Limited Block Query Protocol (*LMTBLKPRC* on page 400)

FIXROWPRC
Fixed Row Query Protocol (*FIXROWPRC* on page 365)

SEE ALSO

Variable	Reference
semantic	*FIXROWPRC* on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYBLKSZ on page 559

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 or LMTBLKPRC.

The value FRCFIXROW means that the target SQLAM must use the fixed row query protocol for all database cursors (see FRCFIXROW).

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2132'	
value	INSTANCE_OF	CODPNTDR - Code Point 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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	OPNQRY on page 475
semantic	*BGNBND* on page 101
	ENDBND on page 307
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	REBIND on page 606

NAME

QRYBLKSZ — Query Block Size

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2114'

Length *

Class CLASS

Sprcls BIN - Binary Integer Number

Query Block Size (QRYBLKSZ) expresses the query block size that the requester wants to receive the query answer set description and data. See the description of the term QRYBLK for a definition of a query block.

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'2114'	
value	INSTANCE_OF LENGTH MINVAL MAXVAL REQUIRED	UNSBINDR - Unsigned Binary Data Representation 32 512 32767
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475
semantic	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475
	QRYBLK on page 555

QRYDSC

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. Multiple QRYDSC objects can be returned to describe the answer set data for a single query. If multiple QRYDSC objects are returned for a single query, the second QRYDSC (and any subsequent ones) are concatenated to the previous QRYDSC to form a complete FD:OCA description of the answer set data.[20]

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'241A'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*EXCSQLSTT* on page 336
	OPNQRY on page 475
semantic	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYBLK on page 555
	QRYDTA on page 561

20. The FD:OCA description for the data value of this object is described in the DRDA Reference.

NAME

QRYDTA — Query Answer Set Data

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'241B'

Length *

Class CLASS

Sprcls FDODTA - 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'241B'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475
semantic	*CNTQRY* on page 217
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	QRYBLK on page 555

QRYNOPRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2202'	
svrcod	INSTANCE_OF ENUVAL ENUVAL REQUIRED	SVRCOD - Severity Code 4 - WARNING - Warning Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*CLSQRY* on page 163
	CNTQRY on page 217
semantic	*CLSQRY* on page 163
	CNTQRY on page 217
	FIXROWPRC on page 365

Variable	Reference
	LMTBLKPRC on page 400

QRYPOPRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'220F'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*OPNQRY* on page 475
semantic	*CNTQRY* on page 217
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475

Variable	Reference
	QRYBLK on page 555

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2102'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'2418' - FIXROWPRC - Fixed Row Query Protocol
	ENUVAL	X'2417' - LMTBLKPRC - Limited Block Query Protocol
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OPNQRYRM* on page 483

NAME

QRYRELSCR — Query Relative Scrolling Action

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'213C'
Length	*
Class	CLASS
Sprcls	BOOLEAN - Logical Value

Query Relative Scrolling Action (QRYRELSCR) Boolean State controls whether or not relative scrolling action is used for scrollable cursors.

The value TRUE indicates that the scrolling action is relative, and the value FALSE indicates that the scrolling action is absolute; that is, not relative.

QRYRELSCR is used in conjunction with the QRYROWNBR parameter specified on the CNTQRY command.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'213C'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Use a relative scrolling action.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Use an absolute scrolling action
	DFTVAL	X'F1' - TRUE - True State
	NOTE	Use a relative scrolling action.
	REQUIRED	

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
	QRYROWNBR on page 569
semantic	*CNTQRY* on page 217
	QRYROWNBR on page 569

QRYRFRTBL

NAME

QRYRFRTBL — Query Refresh Answer Set Table

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'213E'

Length *

Class CLASS

Sprcls BOOLEAN - Logical Value

Query Refresh Answer Set Table (QRYRFRTBL) Boolean State controls whether the answer set table should be refreshed before returning the row(s) requested for scrollable cursors. Recomputing the query refreshes the answer set table.

The value TRUE indicates that the answer set table should be refreshed.

The value FALSE indicates that the answer set table should not be refreshed.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'213E'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL NOTE REQUIRED	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State Refresh the answer set table. X'F0' - FALSE - False State Do not refresh the answer set table. X'F0' - FALSE - False State The answer set table is not refreshed.

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
semantic	*CNTQRY* on page 217

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.

QRYROWNBR is used in conjunction with the *qryrelscr* parameter specified on a CNTQRY command.

When used with a *qryrelscr* value of FALSE (absolute), it identifies that the row is calculated from the beginning of the answer set table.

When used with a *qryrelscr* value of TRUE (relative), it identifies that the row is calculated from the current row of the answer set table.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'213D'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	NOTE	For a qryrelscr value of TRUE (relative), a value of -N...+N returns the Nth row before (-N) or after (+N) the current row. A value of zero would return the current row.
	DFTVAL	1
	NOTE	Return the next row from the current row.
	NOTE	For a qryrelscr value of FALSE (absolute), a value of +1...+N returns the Nth row.
	DFTVAL	1
	NOTE	Return the first row (first).
	SPCVAL	0
	NOTE	Position before the first row (before).
	SPCVAL	–1
	NOTE	Return the last row (last).
	SPCVAL	–2
	NOTE	Position after the last row (after).
	OPTIONAL	

SEE ALSO

Variable	Reference
insvar	*CNTQRY* on page 217
semantic	*CNTQRY* on page 217
	QRYRELSCR on page 567

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 the description of the term SQL 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 the term *RDBOVR* on page 593 for a discussion on how DDM facilities access a remote RDB.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'240F'	
typdef	INSTANCE_OF	TYPDEF - Data Type to Data Representation Definition
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	3	
mgrdepls	**NIL**	
vldattls		**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

Variable	Reference
insvar	*DECPRC* on page 261
	DFTRDBCOL on page 268
	DGRIOPRL on page 270
	MGRLVL on page 427
	RDBALWUPD on page 580
	RDBCMTOK on page 585
	STTSTRDEL on page 744

Variable	Reference
	TRGNSPRM on page 868
mgrdepls	*SQLAM* on page 694
semantic	*ABBREVIATIONS* on page 30
	ABNUOWRM on page 39
	ACCDMG on page 41
	ACCRDB on page 42
	ACCRDBRM on page 48
	AGNCMDPR on page 60
	BGNBND on page 101
	BNDCHKEXS on page 119
	BNDCHKONL on page 120
	BNDERRALW on page 122
	BNDEXPOPT on page 123
	BNDEXSOPT on page 124
	BNDEXSRQR on page 125
	BNDNERALW on page 126
	BNDSQLSTT on page 130
	BNDSTTASM on page 135
	CLSQRY on page 163
	CMNAPPC on page 179
	CMNLYR on page 189
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	CNTQRY on page 217
	CSTBITS on page 241
	CSTMBCS on page 242
	CSTSBCS on page 243
	CSTSYSDFT on page 244
	DFTRDBCOL on page 268
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DTAOVR on page 305

Variable	Reference
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	EXPALL on page 344
	EXPNON on page 345
	FDOCA on page 351
	FIXROWPRC on page 365
	INHERITANCE on page 380
	ISOLVLNC on page 393
	LMTBLKPRC on page 400
	MANAGER on page 417
	MGROVR on page 436
	OPNQRY on page 475
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGCNSTKN on page 500
	PKGID on page 504
	PKGISOLVL on page 505
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGOWNID on page 512
	PKGSN on page 519
	PKGSNLST on page 520
	PRPSQLSTT on page 533
	RDBACCCL on page 577
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBALWUPD on page 580
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBCMTOK on page 585
	RDBCOLID on page 586
	RDBNACRM on page 587
	RDBNAM on page 589

Variable	Reference
	RDBNFNRM on page 592
	RDBOVR on page 593
	RDBRLLBCK on page 598
	RDBRLSCMM on page 601
	RDBRLSCNV on page 602
	RDBRLSOPT on page 603
	RDBUPDRM on page 604
	REBIND on page 606
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SQL on page 680
	SQLAM on page 694
	SQLCARD on page 702
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLOBJNAM on page 712
	SQLSTT on page 714
	SRVLST on page 725
	SUBSETS on page 747
	SYNCCTL on page 760
	SYNCPTMGR on page 782
	SYNCPTOV on page 787
title	*ACCRDB* on page 42
	ACCRDBRM on page 48
	BGNBND on page 101
	BNDSQLSTT on page 130
	DFTRDBCOL on page 268
	DRPPKG on page 274
	DSCRDBTBL on page 281
	ENDBND on page 307
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGCNSTKN on page 500
	PKGID on page 504
	PKGNAM on page 507

Variable	Reference
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PKGSN on page 519
	PKGSNLST on page 520
	RDBACCCL on page 577
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBALWUPD on page 580
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBCMTOK on page 585
	RDBCOLID on page 586
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBRLLBCK on page 598
	RDBRLSCMM on page 601
	RDBRLSCNV on page 602
	RDBRLSOPT on page 603
	RDBUPDRM on page 604
	REBIND on page 606
	RSLSETRM on page 642

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'210F'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	REQUIRED	
	ENUVAL	X'2407' - SQLAM - SQL Application Manager
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
semantic	*ACCRDB* on page 42
	AGNCMDPR on page 60

RDBACCRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2207'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
semantic	*ACCRDB* on page 42

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'221A'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
rpydta	*ACCRDB* on page 42
semantic	*ACCRDB* on page 42

RDBALWUPD

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.

clsvar	**NIL**	
insvar		**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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
semantic	*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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2203'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
semantic	*ACCRDB* on page 42
	AGNCMDPR on page 60

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.

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.

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 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.

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'200E'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*CMDVLTRM* on page 176
	OPNQRY on page 475
	SQL on page 680
	SQLAM on page 694
	SUBSETS on page 747

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'2105'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State RDB should allow the processing of commits and rollbacks. X'F0' - FALSE - False State RDB should not allow the processing of commits and rollbacks. X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336
semantic	*EXCSQLIMM* on page 331
	EXCSQLSTT on page 336

RDBCOLID

NAME

RDBCOLID — RDB Collection Identifier

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'2108'
Length	*
Class	CLASS
Sprcls	FIELD - A Discrete Unit of Data

RDB Collection Identifier (RDBCOLID) identifies a unique collection of objects contained in a relational database, and it is used for user-defined grouping.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
name	INSTANCE_OF LENGTH	NAMSYMDR - Name Symbol Data Representation 18
clscmd	**NIL**	
inscmd	**NIL**	
vldattls	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PKGNAM* on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
semantic	*PKGID* on page 504
	SQL on page 680

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2204'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331

Variable	Reference
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
semantic	*AGNCMDPR* on page 60
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	22	
class	X'2110'	
value	INSTANCE_OF	NAMSYMDR - Name Symbol Data Representation
	LENGTH	18
	REQUIRED	
	NOTE	The syntax of this field is not validated.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ABNUOWRM* on page 39
	ACCRDB on page 42
	AGNPRMRM on page 61
	BGNBND on page 101
	BGNBNDRM on page 109
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177

Variable	Reference
	CNTQRY on page 217
	DRPPKG on page 274
	DSCINVRM on page 279
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DTAMCHRM on page 303
	ENDBND on page 307
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	MGRDEPRM on page 426
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRY on page 475
	PKGBNARM on page 497
	PKGBPARM on page 498
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	PRPSQLSTT on page 533
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBCMM on page 582
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBRLLBCK on page 598
	RDBUPDRM on page 604

Variable	Reference
	REBIND on page 606
	RSCLMTRM on page 634
	SQLERRRM on page 710
	SYNCLOG on page 764
	SYNTAXRM on page 835
	TRGNSPRM on page 868
	VALNSPRM on page 898
mgrdepls	*SQLAM* on page 694
semantic	*ACCRDB* on page 42
	AGNCMDPR on page 60
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SQL on page 680
	SRVLST on page 725
	SUBSETS on page 747
vldattls	*RDB* on page 571

RDBNFNRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2211'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
semantic	*ACCRDB* on page 42
	AGNCMDPR on page 60

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 the description of the term SQL 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 3-65 on page 594.

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.

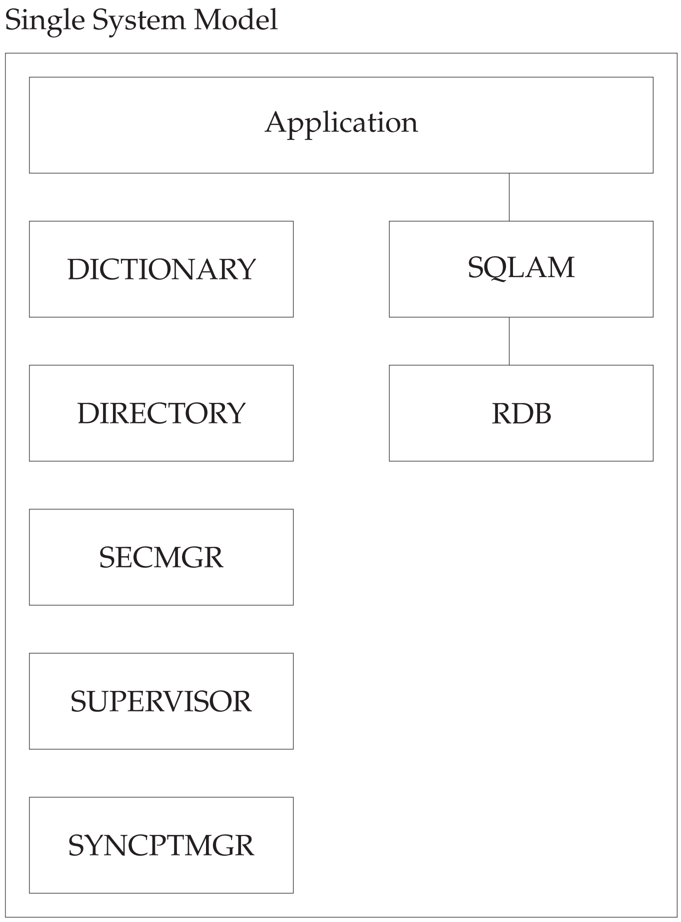


Figure 3-65 Application to Local RDB Connection

Distributed Systems Model

The DDM architecture extends the single-system model in Figure 3-65 to the distributed systems model in Figure 3-66 on page 595 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.

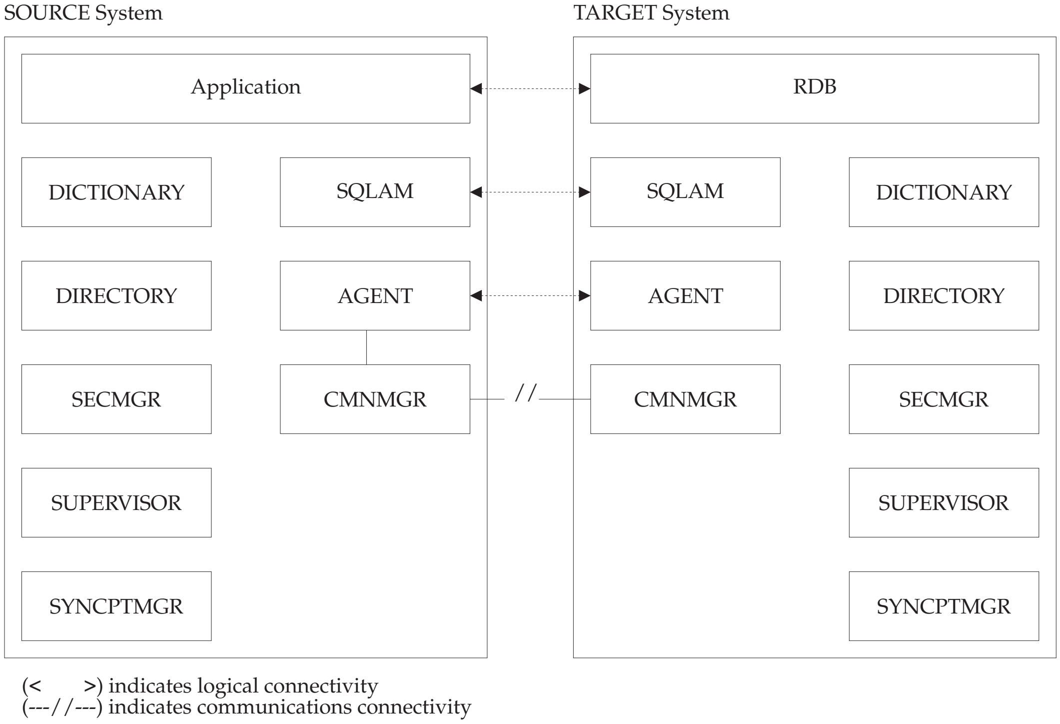


Figure 3-66 Application to Remote RDB Connection

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:

   1. The instantiation of an SQLAM for the application to use
   2. How an SQLAM is instantiated for the application to use
   3. The mechanisms used to transmit messages between the application and the SQLAM
   4. The translation of native source system interfaces to DDM commands and objects
   5. The translation of DDM reply messages and objects to native source system interfaces

2. The source SQLAM performs the following services:

   1. 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.
   2. 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.
   3. The source SQLAM converts data between the application and the RDB data representations for all types of SQL data.

   4. When required, the source SQLAM converts data from the target RDB to the application data representations for all instances of SQL data types.
   5. When required, the source SQLAM interacts with the source sync point manager (described in the term SYNCPTMGR) for two-phase commit and backout processing.
3. The source agent (described in the term AGENT) performs the following services:
   1. Routes commands and command data received from the source SQLAM to a source communications manager (described in the term CMMMGR)
   2. Parses and validates reply messages and reply data received from the source communications manager and routes them to the source SQLAM
   3. 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 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 CMNSYNCPT, 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

Variable	Reference
semantic	*MGROVR* on page 436
	RDB on page 571
	SQL on page 680
	SQLAM on page 694

RDBRLLBCK

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.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'200F'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
rpydta		**REPLY OBJECTS**
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	TYPDEFOVR - TYPDEF Overrides ''

	NOTE	The default means the value received on ACCRDBRM is used.
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*CMDVLTRM* on page 176
	OPNQRY on page 475
	SQL on page 680
	SQLAM on page 694
	SUBSETS on page 747

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

Variable	Reference
insvar	*RDBRLSOPT* on page 603
semantic	*ENDBND* on page 307

RDBRLSCNV

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

Variable	Reference
insvar	*RDBRLSOPT* on page 603

NAME

RDBRLSOPT — RDB Release Option

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2129'

Length *

Class CLASS

Sprcls STRING - String

RDB Release Option (RDBRLSOPT) 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2129'	
value	INSTANCE_OF	CODPNTDR - Code Point 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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	REBIND on page 606
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606

RDBUPDRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2218'	
svrcod	INSTANCE_OF ENUVAL REQUIRED	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*BGNBND* on page 101
	BNDSQLSTT on page 130
	DRPPKG on page 274
	ENDBND on page 307

Variable	Reference
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	REBIND on page 606
semantic	*BGNBND* on page 101
	BNDSQLSTT on page 130
	DRPPKG on page 274
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	ISOLVLNC on page 393
	REBIND on page 606
	SQLAM on page 694
	SYNCPTOV on page 787

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 *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 *pkgnam* parameter specifies the fully qualified name of the existing package which is rebinding.

The *vrsnam* parameter specifies the version name associated with the named existing 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 *bndexpopt* parameter controls whether the target SQLAM causes the target RDB to explain all explainable SQL statements bound into the package.

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

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

Some items of the existing package cannot be changed during the rebind process. These are:

- The fully qualified package name (PKGNAM)
- The package consistency token (PKGCNSTKN)
- The package version name (VRSNAM)
- 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2010'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
pkgnam	INSTANCE_OF REQUIRED	PKGNAM - RDB Package Name
vrsnam	INSTANCE_OF OPTIONAL	VRSNAM - Version Name
pkgisolvl	INSTANCE_OF OPTIONAL DFTVAL NOTE	PKGISOLVL - Package Isolation Level '' Defaults to the value specified for the existing package.
bndexpopt	INSTANCE_OF OPTIONAL DFTVAL NOTE	BNDEXPOPT - Bind Explain Option '' Defaults to the value specified for the existing package.
bndchkexs	INSTANCE_OF OPTIONAL DFTVAL NOTE	BNDCHKEXS - Bind Existence Checking '' Defaults to the value specified for the

		existing package.
dftrdbcol	INSTANCE_OF	DFTRDBCOL - Default RDB Collection ID
	OPTIONAL	
	DFTVAL	''
	NOTE	Defaults to the value specified for the existing package.
dgrioprl	INSTANCE_OF	DGRIOPRL - Degree of I/O Parallelism
	OPTIONAL	
	DFTVAL	''
	NOTE	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	PKGATHRUL - Package Authorization Rules
	OPTIONAL	
	DFTVAL	''
	NOTE	Defaults to the value specified for the existing package.
	IGNORABLE	
	MINLVL	5
pkgownid	INSTANCE_OF	PKGOWNID - Package Owner Identifier
	OPTIONAL	
	DFTVAL	''
	NOTE	Defaults to the value specified for the existing package.
rdbrlsopt	INSTANCE_OF	RDBRLSOPT - RDB Release Option
	OPTIONAL	
	DFTVAL	''
	NOTE	Defaults to the value specified for the existing package.
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'2405'	INSTANCE_OF	BNDOPT - Bind Option
	OPTIONAL	
	REPEATABLE	
	MINLVL	5
rpydta		**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

	OPTIONAL	
	DFTVAL	''
	NOTE	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'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
cmdrpy		**COMMAND REPLIES**
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	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

Variable	Reference
inscmd	*SQLAM* on page 694
semantic	*BGNBND* on page 101
	BNDCHKEXS on page 119
	BNDEXSRQR on page 125
	SQL on page 680

Variable	Reference
	SQLAM on page 694

REPEATABLE

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0031'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*BGNBND* on page 101
	BNDSQLSTT on page 130
	REBIND on page 606
	SYNCCTL on page 760
insvar	*ARRAY* on page 96
	ATTLST on page 99
	BITSTRDR on page 117
	CHRSTRDR on page 146
	CODPNT on page 225
	CONSTANT on page 238
	DCTIND on page 253
	DEFLST on page 263
	FDOOBJ on page 360

Variable	Reference
	HELP on page 371
	HEXSTRDR on page 376
	MGRLVLLS on page 429
	NAMSYMDR on page 447
	PKGSNLST on page 520
	SECMEC on page 661
	SQLDTA on page 708
	SQLDTARD on page 709
	SRVLST on page 725
rpydta	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	OPNQRY on page 475
semantic	*ENUCLS* on page 316

REQUESTER

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

Variable	Reference
semantic	*PKGATHRUL* on page 494
	PKGATHRUL on page 494

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 the description of the term SUBSETS.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0032'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*SECCHK* on page 652
insvar	*EXCSQLSTT* on page 336
	PRCNAM on page 525
	SECCHK on page 652
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	COMMAND on page 233
	LVLCMP on page 412
	MGRLVLN on page 430
	SECCHK on page 652
	SUBSETS on page 747

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0033'	
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'.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
	DSSFMT on page 301
	RSLSETFLG on page 639

RESYNOVR

NAME

RESYNOVR — Resynchronization Overview

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length *

Class HELP

Resynchronization Overview (RESYNOVR) 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

Variable	Reference
semantic	*CMNSYNCPT* on page 197
	SYNCMNFL on page 771

NAME

RLSCONV — Release Conversation

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'119F'

Length *

Class CLASS

Sprcls BOOLEAN - Logical Value

Release Conversation (RLSCONV) specifies whether a conversation should be terminated 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'119F'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State X'F1' - Terminate released conversation. X'F0' - FALSE - False State X'F0' - Do not terminate connection. X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCCTL* on page 760

RPYDSS

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 3-67 and in the *nsvar* 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 3-67 maps a reply message onto a data stream.

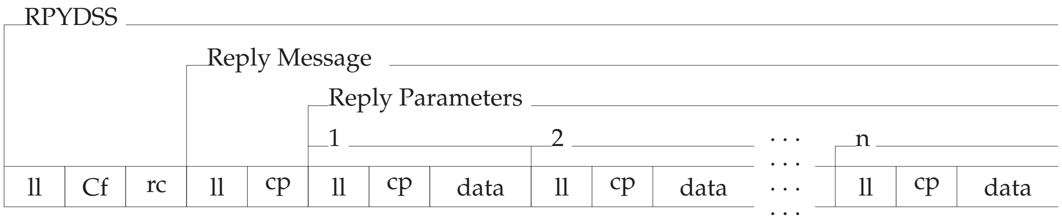


Figure 3-67 Reply Data Stream Structures

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 Code point of an object

data Value of an object

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	INSTANCE_OF LENGTH MINVAL	BINDR - Binary Number Field 16 6

	MAXVAL	32767
	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
	REQUIRED	that request.
data	MINLEN	0
	MAXLEN	32761
	NOTE	Instances of a defined class of reply messages are mapped into this data field. The code point of the message identifies a specific reply message (for instance 120B identifies the ENDFILRM).
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PRCCNVCD* on page 521
semantic	*ABNUOWRM* on page 39
	ACCRDBRM on page 48
	AGNPRMRM on page 61
	AGNRPYPR on page 63
	APPCMNFL on page 64
	APPCMNI on page 68
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	BGNBNDRM on page 109
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	COMMAND on page 233
	DSCINVRM on page 279
	DSS on page 289
	DTAMCHRM on page 303
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	INHERITANCE on page 380
	LMTBLKPRC on page 400
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJNSPRM on page 462
	OPNQFLRM on page 474

Variable	Reference
	OPNQRYRM on page 483
	PKGBNARM on page 497
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	QRYBLK on page 555
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBOVR on page 593
	RDBUPDRM on page 604
	RPYMSG on page 624
	RQSCRR on page 626
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SECCHKRM on page 659
	SQLERRRM on page 710
	SYNCMNFL on page 771
	SYNCMNI on page 774
	SYNTAXRM on page 835
	TCPCMNFL on page 838
	TCPCMNI on page 840
	TCPSRCCD on page 853
	TCPSRCER on page 859
	TRGNSPRM on page 868
	VALNSPRM on page 898

RPYMSG

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1437'	
svrcod	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*DSS* on page 289
	INHERITANCE on page 380
sprcls	*ABNUOWRM* on page 39
	ACCRDBRM on page 48
	AGNPRMRM on page 61
	BGNBNDRM on page 109
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172

Variable	Reference
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	DSCINVRM on page 279
	DTAMCHRM on page 303
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRYRM on page 483
	PKGBNARM on page 497
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBUPDRM on page 604
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SECCHKRM on page 659
	SQLERRRM on page 710
	SYNTAXRM on page 835
	TRGNSPRM on page 868
	VALNSPRM on page 898

RQSCRR

NAME

RQSCRR — Request Correlation Identifier

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1438'

Length *

Class CLASS

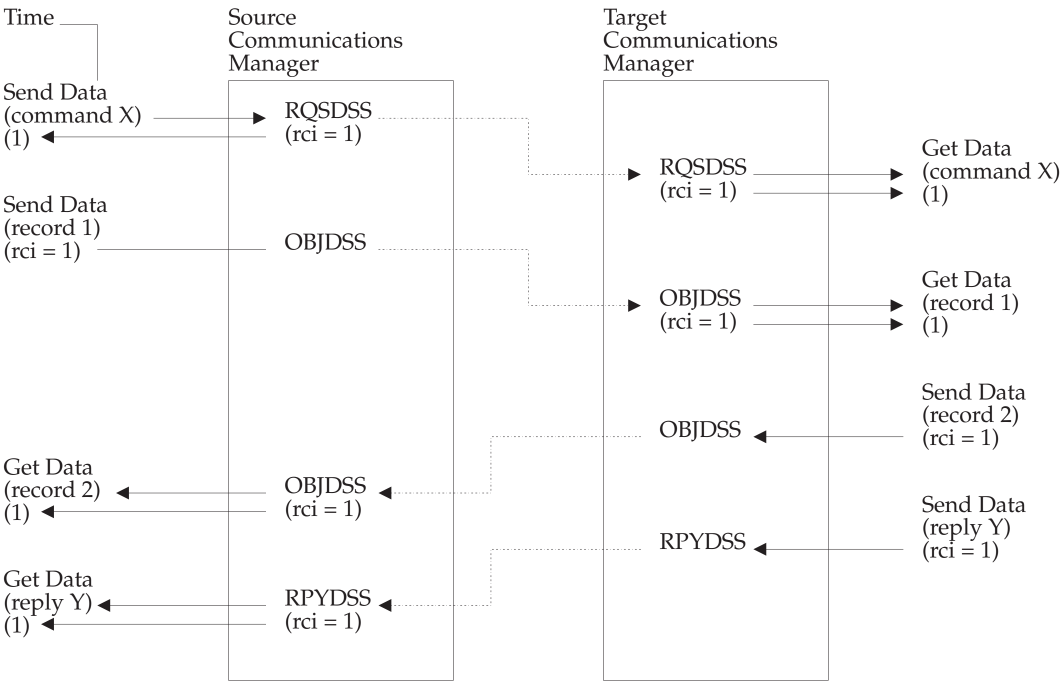
Sprcls DATA - Encoded Information

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 3-68 on page 627 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 3-68 Data Stream Structure Request Correlation

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
value	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OBJDSS* on page 455
	RPYDSS on page 620

Variable	Reference
	RQSDSS on page 629
semantic	*AGNCMDPR* on page 60
	AGNRPYPR on page 63
	DSS on page 289
	SUBSETS on page 747

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 3-69 and in the *nsvar* 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 3-69 maps a command onto the data stream.

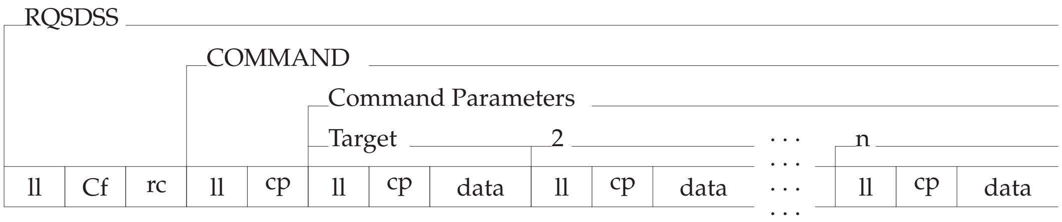


Figure 3-69 Request Data Stream Structures

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 Code point of an object

data Value of an object

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	INSTANCE_OF LENGTH MINVAL MAXVAL	BINDR - Binary Number Field 16 6 32767

	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	DDMID - DDM Identifier
	REQUIRED	
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	5
	REQUIRED	
rqscrr	INSTANCE_OF	RQSCRR - Request Correlation Identifier
	NOTE	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 code point of the command identifies the class of the command.
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OBJDSS* on page 455
	PRCCNVCD on page 521
	SYNERRCD on page 831
semantic	*ACCRDB* on page 42
	ACCSEC on page 51
	AGNCMDPR on page 60
	APPCMNFL on page 64
	APPCMNI on page 68
	APPCMNT on page 72
	APPSRCCD on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLASS on page 148

Variable	Reference
	CLSQRY on page 163
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	DSS on page 289
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	INHERITANCE on page 380
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBOVR on page 593
	RDBRLLBCK on page 598
	REBIND on page 606
	RQSCRR on page 626
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCMNI on page 774
	SYNCMNT on page 778
	SYNCRSY on page 828
	TCPCMNFL on page 838
	TCPCMNI on page 840
	TCPSRCCD on page 853
	TCPSRCCR on page 856

Variable	Reference
	TCPSRCER on page 859
	TCPTRGER on page 861

RSCLMTRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1233'	
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
csrposst	INSTANCE_OF	CSRPOSST - Cursor Position Status
	OPTIONAL	
	NOTE	This parameter is not returned by commands that operate on RDBs.
	MINLVL	3

dclnam	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
dtalckst	INSTANCE_OF OPTIONAL NOTE MINLVL	DTALCKST - Data Lock Status This parameter is not returned by commands that operate on RDBs. 3
filnam	INSTANCE_OF OPTIONAL NOTE NOTE MINLVL	FILNAM - File Name Required when FILNAM was specified on the command. This parameter is not returned by commands that operate on RDBs. 3
drcnam	INSTANCE_OF MINLVL OPTIONAL NOTE NOTE MINLVL	DRCNAM - Directory Name 2 Required when DRCNAM was specified on the command. This parameter is not returned by commands that operate on RDBs. 3
rdbnam	INSTANCE_OF MINLVL OPTIONAL NOTE NOTE	RDBNAM - Relational Database Name 3 Required when RDBNAM was specified on the command. This parameter is not returned by commands that operate on files.
rscnam	INSTANCE_OF MINLVL OPTIONAL	RSCNAM - Resource Name Information 3
rsctyp	INSTANCE_OF MINLVL OPTIONAL	RSCTYP - Resource Type Information 3
prdid	INSTANCE_OF MINLVL OPTIONAL	PRDID - Product-Specific Identifier 3
rsncod	INSTANCE_OF MINLVL OPTIONAL	RSNCOD - Reason Code Information 3

reccnt	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
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
semantic	*DSS* on page 289
	RSCTYP on page 638

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'112D'	
value	INSTANCE_OF	CHRSTRDR - Character String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RSCLMTRM* on page 634

RSCTYP

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'111F'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'1409' - STGLMT - Storage Limit Reached
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RSCLMTRM* on page 634

NAME

RSLSETFLG — Result Set Flags

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2142'

Length *

Class CLASS

Sprcls BITSTRDR - Bit String Field

Result Set Flags specifies whether the requester desires the target SQLAM to return name, label, and comment information for the columns of result sets generated by an EXCSQLSTT command that invokes a stored procedure and whether the requester desires the target SQLAM to return answer set data for the result sets in the response to the EXCSQLSTT. Each bit is a boolean flag.

0 *names* means that the requester requires the target SQLAM to return the names for the columns of result sets in order to ensure the correct operation of the application invoking the stored procedure.

- **0** Zero means that the requester does not desire the target SQLAM to return the names for columns contained within result sets.
- **1** One means that the requester requires the target SQLAM to return the names for columns contained within result sets.

1 *labels* means that the requester requires the target SQLAM to return the labels for the columns of result sets in order to ensure the correct operation of the application invoking the stored procedure.

- **0** Zero means that the requester does not desire the target SQLAM to return the labels for columns contained within result sets.
- **1** One means that the requester requires the target SQLAM to return the labels for columns contained within result sets.

2 *comments* means that the requester requires the target SQLAM to return the comments for the columns of result sets in order to ensure the correct operation of the application invoking the stored procedure.

- **0** Zero means that the requester does not desire the target SQLAM to return the comments for columns contained within result sets.
- **1** One means that the requester requires the target SQLAM to return the comments for columns contained within result sets.

3 *dsconly* means that 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.

- **0** Zero means that, for each result set, the requester expects the command to return an FD:OCA description of the answer set data and to possibly return answer set data. The block containing the end of the FD:OCA description may be completed, if room exists, with answer set data. Additional blocks of answer set data may also be chained to the block containing the end of the

FD:OCA description, up to the maximum number of extra blocks of answer set data per result set specified in the MAXBLKEXT parameter.

1 One means that, for each result set, the requester expects the command to return an FD:OCA description of the answer set data but does not expect the command to return any answer set data.

4-7 Reserved.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'2142'	
names	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	BITDR - A Single Bit 1 B'0' Column names are not desired. B'1' Column names are required.
labels	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	BITDR - A Single Bit 1 B'0' Column labels are not desired. B'1' Column labels are required.
comments	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	BITDR - A Single Bit 1 B'0' Column comments are not desired. B'1' Column comments are required.
dsconly	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	BITDR - A Single Bit 1 B'0' The requester is capable of processing answer set data returned in the response to EXCSQLSTT. B'1' The requester is not capable of processing answer set data returned in the response to EXCSQLSTT.
rest	INSTANCE_OF LENGTH RESERVED	BITDR - A Single Bit 4
default	DFTVAL OPTIONAL	B'00000000'

clscmd	**NIL**
inscmd	**NIL**

SEE ALSO

Variable	Reference
insvar	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2219'	
svrcod	INSTANCE_OF ENUVAL REQUIRED	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
pkgsnlst	INSTANCE_OF REQUIRED	PKGSNLST - RDB Package Name, Consistency Token, and Section Number
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*EXCSQLSTT* on page 336
semantic	*CNTQRY* on page 217
	EXCSQLSTT on page 336
	LMTBLKPRC on page 400
	QRYBLK on page 555

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'1127'	
value	INSTANCE_OF LENGTH	BYTSTRDR - Byte String 4
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RSCLMTRM* on page 634

RSYNCMGR

NAME

RSYNCMGR — Resynchronization Manager

DESCRIPTION (Semantic)

Dictionary	QDDBASD
Codepoint	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.

The role of the RSYNCMGR at DDM Level 5 is illustrated in Figure 3-70 on page 645. 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.

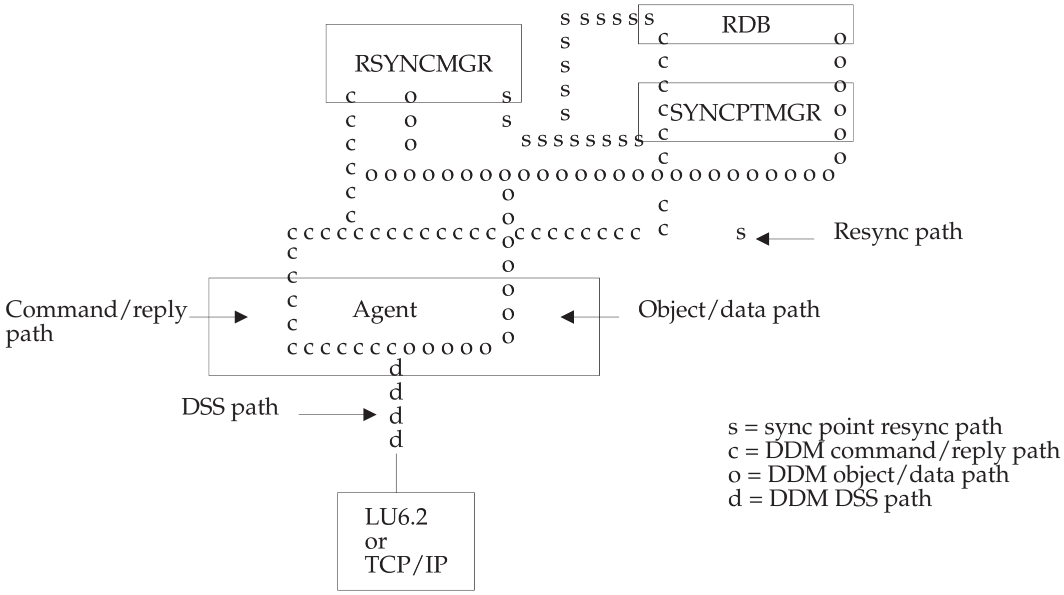


Figure 3-70 Server Paths for RSYNCMGR at DDM Level 5

The sync point manager (SYNCPTMGR) interfaces with the AGENT to perform resynchronization.

Manager-Level Compatibility

Table 3-14 illustrates the function of the SYNCPTMGR as it has grown and changed through the levels of DDM architecture.

Table 3-14 Sync Point Manager-Level Compatibility

DDM Levels	1	2	3	4	5
RSYNCMGR					5
Manager Dependencies					
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 help term SYNCPTOV.

clsvar	**NIL**
insvar	**NIL**
clscmd	**NIL**

inscmd	**NIL**	
mgrlvln	4	
mgrdepls		**MANAGER DEPENDENCY LIST**
X'1444'	INSTANCE_OF MGRLVLN	CMNAPPC - LU 6.2 Conversational Communications Manager 3
X'1474'	INSTANCE_OF MGRLVLN	CMNTCPIP - TCP/IP Communication Manager 5
X'14C0'	INSTANCE_OF MGRLVLN	SYNCPTMGR - Sync Point Manager 5
X'1403'	INSTANCE_OF MGRLVLN	AGENT - Agent 5
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

None.

NAME

RSYNCTYP — Resync Type

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint 11EA

Length *

Class CLASS

Sprcls NAME - Name

Resync Type (RSYNCTYP) specifies the type of resync request or reply data.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*SYNCRSY* on page 828
insvar	*SYNCRRD* on page 826
	SYNCRSY on page 828
semantic	*SYNCRRD* on page 826

RTNSQLDA

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'2116'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State Return the SQLDA. X'F0' - FALSE - False State Do not return the SQLDA. X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*PRPSQLSTT* on page 533
rpydta	*PRPSQLSTT* on page 533

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 MGRLVL which must be encoded *codpnt, mgrlvln*.

All scalars are mapped onto the DDM data stream as shown in Figure 3-71.

length	code point	scalar data

Figure 3-71 Scalar to Data Stream Mapping

- Length is the total length of the scalar, including the length, code point, and data fields.

  See the description of the term DSS 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
- Code point 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 code point 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 *.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	4	
class	X'0034'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*COLLECTION* on page 231
	DSS on page 289
	INHERITANCE on page 380
	OBJECT on page 459
	OBJOVR on page 464
sprcls	*BOOLEAN* on page 136
	CMMTYP on page 178
	CNSVAL on page 216
	DFTVAL on page 269
	ENUVAL on page 318
	FDODTA on page 357
	IPADDR on page 388
	MAXVAL on page 425
	MGRLVLLS on page 429
	MINVAL on page 441
	PKGNAM on page 507
	PKGNAMCSN on page 508
	PKGNAMCT on page 510
	RESERVED on page 617
	SNAADDR on page 674
	SPCVAL on page 676
	SRVLCNT on page 722
	SRVPRTY on page 730
	SYNCTYPE on page 830
	TCPHOST on page 846
	UOWDSP on page 882

Variable	Reference
	UOWID on page 883
	UOWSTATE on page 885

SECCHK

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 3-15shows how the selected security mechanisms control the parameters or data objects sent on the SECCHK command.

Table 3-15 SECCHK Valid Security Mechanism Combinations

DCESEC	USRIDPWD	USRIDONL	USRIDNWPWD	
SECTKN	X	X		
USRID		X	X	X
PASSWORD		X		X
NEWPASSWORD				X

SECMEC and SECTKN Rules

If the SECMEC value is DCESEC 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. (See DDM terms DCESECOVR for more 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

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.

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'106E'	
secmgrnm	INSTANCE_OF OPTIONAL IGNORABLE NOTE CMDTRG	SECMGRNM - Security Manager Name This parameter has a null value and does not have to be validated.
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.
sectkn	INSTANCE_OF OPTIONAL NOTE IGNORABLE NOTE	SECTKN - Security Token If the security mechanism is DCESEC, NETSPSEC, or USRIDPWD, then the SECTKN is REQUIRED. If the security mechanism is USRIDONL, then the SECTKN is ignored.
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.
newpassword	INSTANCE_OF OPTIONAL NOTE	NEWPASSWORD - New Password If the security mechanism is USRIDNWPWD, then NEWPASSWORD is REQUIRED.
usrid	INSTANCE_OF OPTIONAL NOTE	USRID - User ID at the Target System USRID is required when the security mechanism is USRIDPWD, USRIDNWPWD, or USRIDONL.
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.
clscmd	**NIL**	
inscmd	**NIL**	
cmddta	**NIL**	
X'11DC'	INSTANCE_OF OPTIONAL NOTE	SECTKN - Security Token If the security mechanism is DCESEC then the SECTKN is REQUIRED.
rpydta	**NIL**	
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.
cmdrpy		**COMMAND REPLIES**
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'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

Variable	Reference
inscmd	*SECMGR* on page 663
insvar	*PRCCNVCD* on page 521
semantic	*ACCSEC* on page 51
	DCESECOVR on page 248
	DCESECTKN on page 252
	SECCHKRM on page 659
	SECMGR on page 663
	SYNCPTOV on page 787
	TCPCMNI on page 840
	USRSECOVR on page 893

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 3-16 shows the relationship between the SECCHKCD parameter and the SVRCOD parameter in the SECCHKRM.

Table 3-16 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'0E'	ERROR
X'0F'	ERROR
X'10'	ERROR
X'12'	ERROR
X'13'	ERROR
X'14'	ERROR
X'15'	ERROR

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	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'0E'
	NOTE	Password expired.
	ENUVAL	X'0F'
	NOTE	Password invalid.
	ENUVAL	X'10'
	NOTE	Password missing.
	ENUVAL	X'12'
	NOTE	Userid missing.
	ENUVAL	X'13'
	NOTE	Userid invalid.
	ENUVAL	X'14'
	NOTE	Userid revoked.
	ENUVAL	X'15'
	NOTE	New Password invalid.
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SECCHKRM* on page 659
semantic	*DCESECOVR* on page 248
	SECCHK on page 652
	SECCHKRM on page 659
	USRSECOVR on page 893

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 the term 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 cmd. 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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1219'	
svrcod	INSTANCE_OF REQUIRED ENUVAL ENUVAL ENUVAL	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code 8 - ERROR - Error Severity Code 16 - SEVERE - Severe Error Severity Code
secchkcd	INSTANCE_OF REQUIRED	SECCHKCD - Security Check Code
sectkn	INSTANCE_OF OPTIONAL IGNORABLE NOTE	SECTKN - Security Token SECTKN is required when the selected security mechanism requires mutual authentication.
svcerrno	INSTANCE_OF NOTE OPTIONAL	SVCERRNO - Security Service Error Number Error number from called service. SRVDGN may contain additional information.

srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*SECCHK* on page 652
semantic	*DCESECOVR* on page 248
	SECCHK on page 652
	SECCHKCD on page 656
	TCPCMNI on page 840
	USRSECOVR on page 893

NAME

SECMEC — Security Mechanism

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'11A2'

Length *

Class CLASS

Sprcls STRING - String

The Security Mechanism (SECMEC) String specifies the security mechanism.

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 Mechanisms

Table 3-17 shows the combinations of the security mechanisms.

Table 3-17 Valid Security Mechanism Combinations

Combination ID	Value	Security Mechanisms
DCESEC	1	OSFDCE
USRIDPWD	3	USRIDSEC and PWDSEC
USRIDONL	4	USRIDSEC
USRIDNWPWD	5	USRIDSEC, PWDSEC, and NWPWDSEC

Combination Rules Summary

The OSFDCE security mechanism must be used by itself.

The USRIDSEC security mechanism can be used with the PWDSEC security mechanism, with the PWDSEC and NWPWDSEC security mechanisms, or just by itself.

The PWDSEC security mechanism cannot be used alone. It requires the USRIDSEC security mechanism.

The NWPWDSEC security mechanism cannot be used alone. It requires the USRIDSEC and PWDSEC security mechanisms.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	

class	X'11A2'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	1 - DCESEC - Distributed Computing Environment Security
	ENUVAL	3 - USRIDPWD - Userid and Password Security Mechanism
	ENUVAL	4 - USRIDONL - Userid Only Security Mechanism
	ENUVAL	5 - USRIDNWPWD - Userid, Password, and New Password Security Mechanism
	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCSEC* on page 51
	ACCSECRD on page 55
	SECCHK on page 652
	SECCHKCD on page 656
semantic	*ACCSEC* on page 51
	DCESECOVR on page 248
	SECCHK on page 652
	SECOVR on page 667
	USRSECOVR on page 893

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.

DCE Security Mechanism

DDM provides support for utilizing the DCE-based security mechanism. See the term *DCESECOVR* on page 248 for more information about the OSF DCE security.

User ID Security and Password Security

See the term *USRSECOVR* on page 893 for the DDM commands and replies that flow in the normal process of establishing a connection while using user ID and password security mechanisms.

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.

Security Manager Level 5 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; if the security commands are not used, then the target server responds with the PRCCNVRM for the command following the EXCSAT command.

Table 3-18 Security Manager-Level Compatibility

DDM Levels	**1**	**2**	**3**	**4**	**5**
SECMGR[22]	1	1	1	1	5
Commands					
ACCSEC					5
SECCHK					5

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1440'	
clscmd	**NIL**	
inscmd		**INSTANCE COMMANDS**
X'106D'	INSTANCE_OF MINLVL REQUIRED	ACCSEC - Access Security 5

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

X'106E'	INSTANCE_OF MINLVL REQUIRED	SECCHK - Security Check 5
mgrlvln	5 NOTE MINLVL	 The SECMGR can be at DDM Level 1 or DDM Level 5. 5
mgrdepls	**NIL**	
vldattls		**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

Variable	Reference
insvar	*DEPERRCD* on page 265
	MGRLVL on page 427
	PRCCNVCD on page 521
	SUPERVISOR on page 753
mgrdepls	*AGENT* on page 56
	SQLAM on page 694
semantic	*ACCSEC* on page 51
	AGENT on page 56
	DCESECOVR on page 248
	DCESECTKN on page 252
	EXTENSIONS on page 346
	INHERITANCE on page 380
	MANAGER on page 417
	NEWPASSWORD on page 450
	PASSWORD on page 491
	RDBOVR on page 593
	SECOVR on page 667
	SQLAM on page 694
	SUBSETS on page 747
	SYNCPTOV on page 787
	USRID on page 888
	USRSECOVR on page 893

SECMGRNM

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	4	
class	X'1196'	
name	LENGTH REQUIRED	0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCSEC* on page 51
	SECCHK on page 652
semantic	*DCESECOVR* on page 248
	USRSECOVR on page 893
vldattls	*SECMGR* on page 663

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.

Definitions

According to *Webster's New Collegiate Dictionary, 1974* and using definition 4(b), security means *measures taken to guard against espionage or sabotage, crime, attack, or escape*. From the *IBM Dictionary of Computing*, ZC20-1699, the following definitions can be found:

1. Computer security means *Concepts, techniques, technical measures, and administrative measures used to protect the hardware, software, and data of an information processing system from deliberate or inadvertent unauthorized acquisition, damage, destruction, disclosure, manipulation, or use, or loss ....*
2. Computer security model means *A mathematical description of the subjects, objects and other entities of a system for the purpose of analyzing the security of the system.*
3. Data processing system security means *The technological and administrative safeguards established and applied to a data processing system to protect hardware, software, and data from accidental or malicious modifications, destruction, or disclosure. Synonymous with computer system security.*
4. Data protection means *The implementation of appropriate administrative, technical, or physical means to guard against unauthorized access to data.*
5. Data security means *The protection of data from accidental or intentional modification or destruction and from accidental or intentional disclosure to unauthorized personnel.*
6. Identity-based access control means *In computer security, access control based on the identities of subjects and the object that are being accessed.*
7. Password means *A value used in authentication or a value used to establish membership in a set of people having specific privileges .... In computer security, a string of characters known to the computer system and a user, who must specify it to gain full or limited access to a system and to the data stored within it.*
8. Password security means *In computer security, the prevention of unauthorized use of a system, device, or program by checking user passwords.*
9. Resource access security (paraphrased) means *In computer security, limiting the resources that can be used by application programs and utilities.*
10. Resource-based access control means *In computer security, access control based on the subject's presentation of evidence of authorization, such as an object-related password, with a request for access to an object.*
11. Sign-on verification security (paraphrased) means *In computer security, identifying a particular user to the computer system.*

12. System security means *A system function that restricts the use of objects to certain users.*
13. Transaction command security (paraphrased) means *In computer security, limiting the system commands that application programs and utilities can issue.*

DDM Architecture and Security

Each DDM server (source and target) uses the security facilities of its system. The Security Manager (see *SECMGR* on page 663) Resource Manager is the logical image of the system security facilities. DDM architecture specifies how to use security. The actual security architecture can be found in the *IBM Security Architecture: A Model for Securing Information Systems*, SC28-8135.

See also the following terms for more information on security:

SECMGR
Security Manager (*SECMGR* on page 663)

SECMEC
Security Mechanism (*SECMEC* on page 661)

SNASECOVR
LU 6.2 Security Overview (*SNASECOVR* on page 675)

DCESECOVR
DCE Security Overview (*DCESECOVR* on page 248)

USRSECOVR
User ID Security Overview (*USRSECOVR* on page 893)

SEE ALSO

Variable	Reference
semantic	*SECTKN* on page 669

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 the term *SECOVR* on page 667 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11DC'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*SECCHK* on page 652
insvar	*SECCHK* on page 652
rpydta	*SECCHK* on page 652
semantic	*ACCSEC* on page 51
	DCESECOVR on page 248
	DCESECTKN on page 252
	SECCHK on page 652
	SECCHKRM on page 659

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1448'	
clscmd	**NIL**	
inscmd	**NIL**	
mgrlvln	1	
mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'116D'	INSTANCE_OF	SRVNAM - Server Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
semantic	*EXCSAT* on page 323
	EXCSATRD on page 329
	INHERITANCE on page 380
	STRLYR on page 737
	SUBSETS on page 747

SESDMG

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

Variable	Reference
insvar	*CMDCHKRM* on page 170
	PRCCNVRM on page 523
	RSCLMTRM on page 634
	SVRCOD on page 756

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

Variable	Reference
insvar	*AGNPRMRM* on page 61
	CMDCHKRM on page 170
	OBJNSPRM on page 462
	PRCCNVRM on page 523
	RSCLMTRM on page 634
	SECCHKRM on page 659
	SVRCOD on page 756

SNAADDR

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SRVLSRV* on page 723
	SYNCLOG on page 764
semantic	*SYNCPTOV* on page 787

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

Variable	Reference
semantic	*SECOVR* on page 667

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0036'	
value	SPRCLS	FIELD - A Discrete Unit of Data
	NOTE	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*DFTVAL* on page 269
	DGRIOPRL on page 270
	MAXBLKEXT on page 420
	MAXRSLCNT on page 423
	MGRLVL on page 427
	QRYROWNBR on page 569
	RQSCRR on page 626
	SRVCLSNM on page 717
mgrlvln	*CCSIDMGR* on page 140

Variable	Reference
semantic	*LVLCMP* on page 412

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'0037'	
value	INSTANCE_OF REQUIRED	CODPNTDR - Code Point Data Representation
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
semantic	*CLASS* on page 148

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	4	
class	X'115D'	
name	INSTANCE_OF ENULEN REQUIRED	BYTSTRDR - Byte String 0
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
vldattls	*SUPERVISOR* on page 753

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:

RDBOVR
Relational Database Overview (*RDBOVR* on page 593)

This term describes the DDM model of remote RDB processing.

RDB Relational Database (*RDB* on page 571)

This term describes the various objects of an RDB.

SQLAM SQL Application Manager (*SQLAM* on page 694)

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 the description of the term *SQLCARD* on page 702 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 the description of the term *SQLCARD* on page 702 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:

RDBNAM
Relational Database Name

PKGNAMCSN
RDB Package Name, Consistency Token, and Section Number

PKGNAMCT
RDB Package Name and Consistency Token

SQL Object Names

SQL statements include references to named objects within an RDB, such as tables, views, and packages (see the description of the term RDB 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

PKGNAMCSN
RDB Package Name, Consistency Token, and Section Number

PKGNAMCT
RDB Package Name and Consistency Token

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:

PKGNAM
RDB Package Name

PKGNAMCSN
RDB Package Name, Consistency Token, and Section Number

PKGNAMCT
RDB Package Name and Consistency Token

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 3-72 on page 685 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.

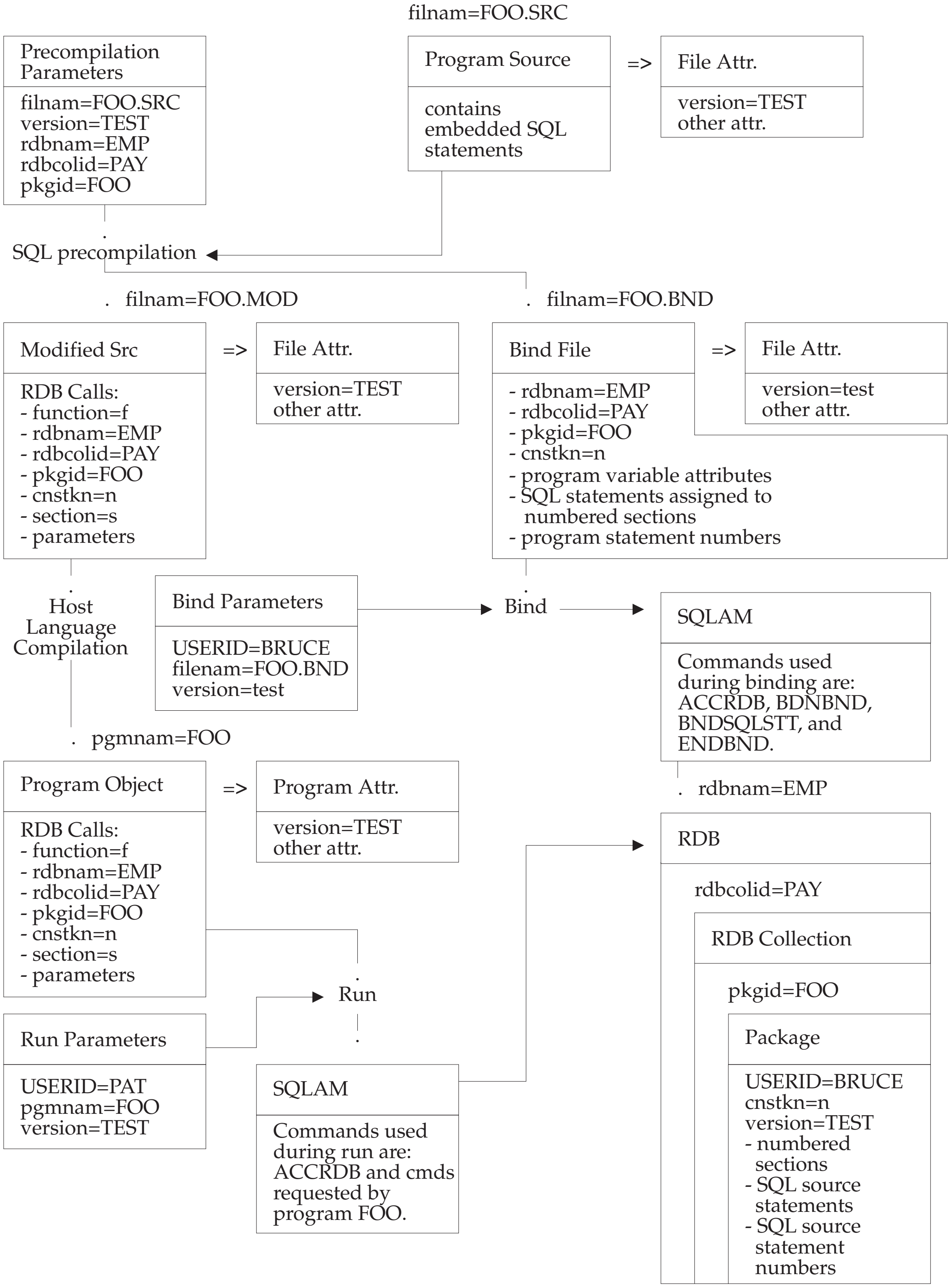


Figure 3-72 Processing Model for Programs with Embedded SQL Statements

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 *local SQL statement*.

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 the term *RDBOVR* on page 593 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 3-72 on page 685 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 the description of the term *RDBOVR* on page 593.

SEE ALSO

Variable	Reference
cmddta	*BNDSQLSTT* on page 130
	EXCSQLSTT on page 336
insvar	*DGRIOPRL* on page 270
	DSCERRCD on page 277
	PKGATHRUL on page 494
	SRVCLSNM on page 717
	STTSTRDEL on page 744
	TYPDEFNAM on page 873
rpydta	*EXCSQLSTT* on page 336
semantic	*ABBREVIATIONS* on page 30
	ACCRDB on page 42
	ACCRDBRM on page 48
	AGENT on page 56
	APPSRCCD on page 75
	BGNBND on page 101
	BNDCHKEXS on page 119
	BNDCHKONL on page 120
	BNDERRALW on page 122
	BNDEXPOPT on page 123
	BNDNERALW on page 126
	BNDSQLSTT on page 130
	BNDSTTASM on page 135
	CMNTCPIP on page 209
	CSTBITS on page 241
	CSTMBCS on page 242
	CSTSBCS on page 243
	DECDELCMA on page 259
	DECDELPRD on page 260
	DFTPKG on page 267
	DFTRDBCOL on page 268
	DGRIOPRL on page 270
	DSCSQLSTT on page 285

Variable	Reference
	DTAOVR on page 305
	ENDBND on page 307
	EURDATFMT on page 321
	EURTIMFMT on page 322
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	EXPALL on page 344
	EXPNON on page 345
	FDOCA on page 351
	FIXROWPRC on page 365
	ISODATFMT on page 389
	ISOLVLALL on page 390
	ISOLVLCHG on page 391
	ISOLVLCS on page 392
	ISOLVLNC on page 393
	ISOLVLRR on page 394
	ISOTIMFMT on page 395
	JISDATFMT on page 396
	JISTIMFMT on page 397
	LMTBLKPRC on page 400
	MGROVR on page 436
	OPNQRY on page 475
	OPNQRYRM on page 483
	OUTEXP on page 489
	OWNER on page 490
	PKGATHRUL on page 494
	PKGDFTCC on page 501
	PKGDFTCST on page 503
	PKGISOLVL on page 505
	PKGOWNID on page 512
	PRPSQLSTT on page 533
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	RDB on page 571
	RDBACCCL on page 577

Variable	Reference
	RDBALWUPD on page 580
	RDBOVR on page 593
	RDBRLSOPT on page 603
	REBIND on page 606
	REQUESTER on page 614
	RTNSQLDA on page 648
	SQLAM on page 694
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLCSRHLD on page 706
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLERRRM on page 710
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714
	SQLSTTNBR on page 715
	SQLSTTVRB on page 716
	STRDELAP on page 733
	STRDELDQ on page 734
	STRLYR on page 737
	STTASMEUI on page 740
	STTDATFMT on page 741
	STTDECDEL on page 742
	STTSCCCLS on page 743
	STTSTRDEL on page 744
	STTTIMFMT on page 746
	SUBSETS on page 747
	TCPSRCCD on page 853
	TYPDEF on page 872
	USADATFMT on page 886
	USATIMFMT on page 887
title	*BNDSQLSTT* on page 130
	BNDSTTASM on page 135

Variable	Reference
	DSCSQLSTT on page 285
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	EXPALL on page 344
	EXPNON on page 345
	PRPSQLSTT on page 533
	RTNSQLDA on page 648
	SQLAM on page 694
	SQLCARD on page 702
	SQLCINRD on page 705
	SQLDTA on page 708
	SQLDTARD on page 709
	SQLERRRM on page 710
	SQLOBJNAM on page 712
	SQLRSLRD on page 713
	SQLSTT on page 714
	SQLSTTNBR on page 715
	SQLSTTVRB on page 716

NAME

SQLAM — SQL Application Manager

DESCRIPTION (Semantic)

Dictionary QDDRDBD

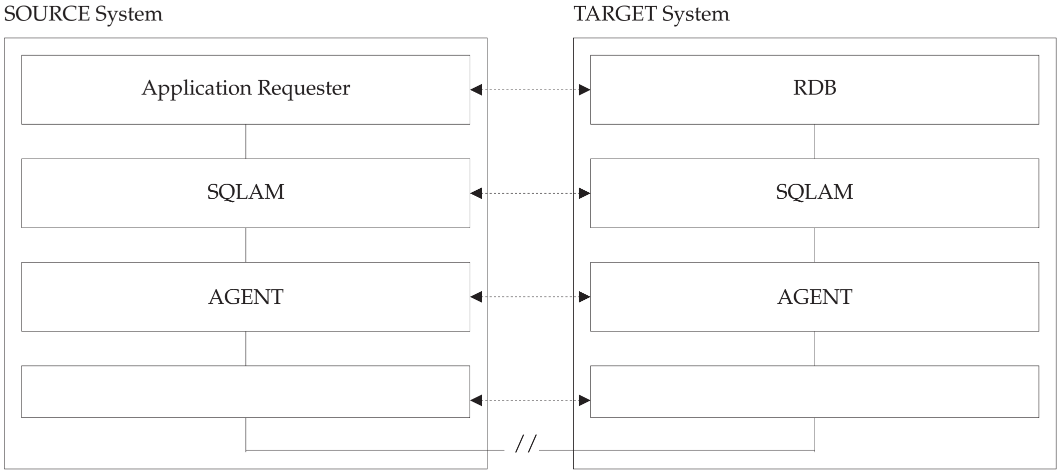
Codepoint X'2407'

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 3-73.



(< >) indicates logical connectivity
(---//---) indicates communications connectivity

Figure 3-73 Application Requester and a Remote RDB

Figure 3-74 on page 695 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 the terms RDBOVR and SQL for overviews.

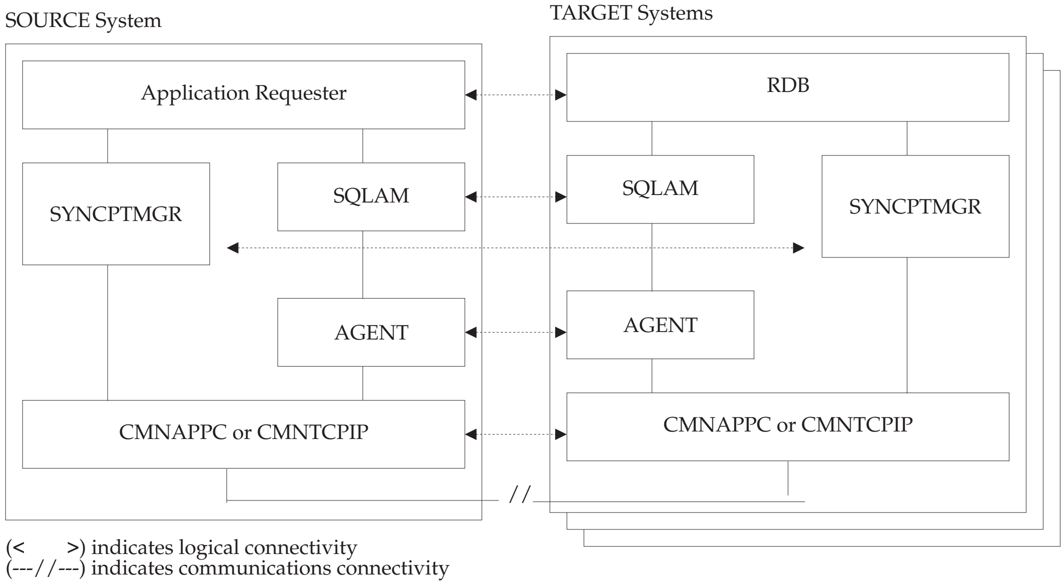


Figure 3-74 Application Requester, Remote RDBs, and Two-Phase Commit

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 (CLSQRY) 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. The description of the term OPNQRY 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 CLSQRY 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 CLSQRY 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 CLSQRY.

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 3-19 illustrates the function of the SQLAM as it has grown and changed through the levels of DDM architecture.

Table 3-19 SQL Application Manager-Level Compatibility

DDM Levels	1	2	3	4	5
SQLAM			3	4	5
Commands					
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
Manager Dependencies					
AGENT			3	4	4
RDB			3	3	3
SECMGR[24]			1	1	5

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2407'	
typdef	INSTANCE_OF	TYPDEF - Data Type to Data Representation Definition
clscmd		**CLASS COMMANDS**
X'2001'	INSTANCE_OF REQUIRED	ACCRDB - Access RDB
inscmd		**INSTANCE COMMANDS**
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	CLSQRY - 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
mgrlvln	5	

mgrdepls		**MANAGER DEPENDENCY LIST**
X'1403'	INSTANCE_OF MGRLVLN 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
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	CLSQRY on page 163
	CNTQRY on page 217
insvar	*ACCRDB* on page 42
	DEPERRCD on page 265
	MAXBLKEXT on page 420
	MAXRSLCNT on page 423
	MGRLVL on page 427
	OUTEXP on page 489
	RDBACCCL on page 577
semantic	*ABNUOWRM* on page 39
	ACCRDB on page 42
	ACCRDBRM on page 48

Variable	Reference
	AGENT on page 56
	AGNCMDPR on page 60
	APPCMNFL on page 64
	BGNBND on page 101
	BNDEXPOPT on page 123
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CMNAPPC on page 179
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	EXPALL on page 344
	EXPNON on page 345
	FIXROWPRC on page 365
	INHERITANCE on page 380
	LMTBLKPRC on page 400
	MANAGER on page 417
	MAXBLKEXT on page 420
	OPNQRY on page 475
	OUTEXP on page 489
	PKGOWNID on page 512
	PRPSQLSTT on page 533
	QRYBLKCTL on page 557
	QRYBLKSZ on page 559
	QRYPRCTYP on page 566
	RDBAFLRM on page 579
	RDBCMM on page 582
	RDBOVR on page 593
	RDBRLLBCK on page 598

Variable	Reference
	REBIND on page 606
	RSCLMTRM on page 634
	RSLSETFLG on page 639
	RTNSQLDA on page 648
	SQL on page 680
	SUBSETS on page 747
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCPTMGR on page 782
	TCPCMNFL on page 838

SQLCARD

NAME

SQLCARD — SQL Communications Area Reply Data

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2408'

Length *

Class CLASS

Sprcls FDODTA - FD:OCA Data

SQL Communications Area Reply Data (SQLCARD) is a byte string that specifies information about conditions detected during relational database (RDB) processing.[25] An SQLCARD contains an SQLCA.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2408'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*ACCRDB* on page 42
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336

25. The FD:OCA description for the data value of this object is described in the DRDA Reference.

Variable	Reference
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
semantic	*ABNUOWRM* on page 39
	ACCRDB on page 42
	BGNBND on page 101
	BGNBNDRM on page 109
	BNDNERALW on page 126
	BNDSQLSTT on page 130
	BNDSTTASM on page 135
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	ENDQRYRM on page 312
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQFLRM on page 474
	OPNQRY on page 475
	PKGRPLNA on page 515
	PRPSQLSTT on page 533
	QRYBLK on page 555
	RDBAFLRM on page 579
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SQL on page 680

Variable	Reference
	SQLERRRM on page 710
	SYNCPTOV on page 787

SQLCINRD

NAME

SQLCINRD — SQL Result Set Column Information Reply Data

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'240B'

Length *

Class CLASS

Sprcls FDODTA - 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.[26]

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'240B'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336
	LMTBLKPRC on page 400

26. The FD:OCA description for the data value of this object is described in the DRDA Reference.

SQLCSRHLD

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X‘211F’	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X‘F1’ - TRUE - True State Specified HOLD option. X‘F0’ - FALSE - False State Did not specify HOLD option. X‘F0’ - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OPNQRYRM* on page 483

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.[27]

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2411'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*DSCRDBTBL* on page 281
	DSCSQLSTT on page 285
	PRPSQLSTT on page 533
semantic	*DSCRDBTBL* on page 281
	DSCSQLSTT on page 285
	PRPSQLSTT on page 533

27. 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.[28]

DSS Carrier: OBJDSS

clsvar	**NIL**	
insvar		**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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*EXCSQLSTT* on page 336
	OPNQRY on page 475
semantic	*EXCSQLSTT* on page 336
	OPNQRY on page 475

28. The FD:OCA description for the data value of this object is described in the DRDA Reference.

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.[29]

DSS Carrier: OBJDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2413'	
fdodsc	INSTANCE_OF REQUIRED REPEATABLE	FDODSC - FD:OCA Data Descriptor
fdodta	INSTANCE_OF OPTIONAL REPEATABLE	FDODTA - FD:OCA Data
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*OUTEXP* on page 489
rpydta	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336
	LMTBLKPRC on page 400
	OUTEXP on page 489
	RSLSETRM on page 642

29. The FD:OCA description for the data value of this object is described in the DRDA Reference.

SQLERRRM

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.

This reply message must precede an SQLCARD object.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2213'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
rdbnam	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*BNDSQLSTT* on page 130
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	PRPSQLSTT on page 533

Variable	Reference
	REBIND on page 606
semantic	*BNDSQLSTT* on page 130

SQLOBJNAM

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.[30] The name can be a one-part, two-part, or three-part RDB object name.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'243E'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 256
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
	DSCRDBTBL on page 281
	DSCRDBTBL on page 281

30. 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	FDODTA - 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.[31]

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'240E'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
rpydta	*EXCSQLSTT* on page 336
semantic	*EXCSQLSTT* on page 336
	LMTBLKPRC on page 400

31. The FD:OCA description for the data value of this object is described in the DRDA Reference.

SQLSTT

NAME

SQLSTT — SQL Statement

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X‘2414’

Length *

Class CLASS

Sprcls FDODTA - FD:OCA Data

SQL Statement (SQLSTT) specifies that an SQL statement is being either executed or bound into an RDB package.[32]

The SQL language standard or product unique extensions, not DDM architecture, defines the syntax and semantics of the SQL statement.

DSS Carrier: OBJDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X‘2414’	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*BNDSQLSTT* on page 130
	EXCSQLIMM on page 331
	PRPSQLSTT on page 533
semantic	*BNDSQLSTT* on page 130
	BNDSTTASM on page 135
	ENDBND on page 307
	EXCSQLIMM on page 331
	PRPSQLSTT on page 533

32. The FD:OCA description for the data value of this object is described in the DRDA Reference.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	8	
class	X'2117'	
value	INSTANCE_OF LENGTH MINVAL REQUIRED	BINDR - Binary Number Field 32 1
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BNDSQLSTT* on page 130
semantic	*BNDSQLSTT* on page 130
	ENDBND on page 307

SQLSTTVRB

NAME

SQLSTTVRB — SQL Statement Variable Descriptions

DESCRIPTION (Semantic)

Dictionary	QDDRDBD
Codepoint	X'2419'
Length	*
Class	CLASS
Sprcls	FDODTA - 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.[33] 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'2419'	
value	INSTANCE_OF REQUIRED	BYTSTRDR - Byte String
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*BNDSQLSTT* on page 130
semantic	*BNDSQLSTT* on page 130
	ENDBND on page 307

33. 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.

SRVCLSNMs beginning with the letter *Q* denote IBM products; while other companies may use any character string to denote their SRVCLSNMs, these SRVCLSNMs cannot begin with the letter *Q*.[34]

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1147'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MAXLEN	255
	SPCVAL	'QAS'
	NOTE	The OS/400 application requester, DDM file server, and database server.
	MINLVL	2
	NOTE	DB2 for OS/400 application requester and application server for relational database.
	MINLVL	5
	SPCVAL	'QDB2'
	NOTE	The DB2 for MVS application requester and application server for relational database.
	MINLVL	3
	SPCVAL	'QSQLDS/VM'
	NOTE	The SQL/DS VM application requester and application server for relational database.
	MINLVL	3
	SPCVAL	'QSQLDS/VSE'
	NOTE	The SQL/DS VSE application requester and application server for relational database.
	MINLVL	3
	SPCVAL	'QDB2/6000'
	NOTE	The DB2 for AIX application requester and

34. See **Trademarks** on page xxi for the list of trademarks referenced in this term.

		application server for relational database.
	MINLVL	4
	SPCVAL	'QDB2/2'
	NOTE	The DB2 for OS/2 application requester and application server for relational database.
	MINLVL	5
	SPCVAL	'QDB2/VM'
	NOTE	The DB2 for VM application requester and application server for relational database.
	MINLVL	5
value (continued)	SPCVAL	'QDB2/VSE'
	NOTE	The DB2 for VSE application requester and application server for relational database.
	MINLVL	5
	SPCVAL	'QDB2/HP-UX'
	NOTE	The DB2 for Hewlett-Packard's HP-UX application requester and application server, for the HP-UX operating system, for relational database.
	MINLVL	5
	SPCVAL	'QDB2/SCO'
	NOTE	The DB2 for SCO application requester and application server, for the Open Server and UNIXware operating system, for relational database.
	MINLVL	5
	SPCVAL	'QDB2/SUN'
	NOTE	The DB2 for SUN application requester and application server, for the Solaris Operating Environment, for relational database.
	MINLVL	5
	SPCVAL	'QDB2/NT'
	NOTE	The DB2 for NT application requester and application server, for the Microsoft Windows NT operating environment, for relational database.
	MINLVL	5
	SPCVAL	'QDB2/SNI'
	NOTE	The DB2 for Siemens Nixdorf SINIX applications requester and applications server.
	MINLVL	5
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
	EXCSATRD on page 329
semantic	*LVLCMP* on page 412
	PRDDTA on page 528

SRVDGN

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 255 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1153'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ABNUOWRM* on page 39
	ACCRDBRM on page 48
	AGNPRMRM on page 61
	BGNBNDRM on page 109
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	CMDNSPRM on page 173
	CMDVLTRM on page 176
	CMMRQSRM on page 177
	DSCINVRM on page 279

Variable	Reference
	DTAMCHRM on page 303
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRYRM on page 483
	PKGBNARM on page 497
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBUPDRM on page 604
	RPYMSG on page 624
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SECCHKRM on page 659
	SQLERRRM on page 710
	SYNERRCD on page 831
	SYNTAXRM on page 835
	TRGNSPRM on page 868
	VALNSPRM on page 898
semantic	*SVCERRNO* on page 755
	USRSECOVR on page 893

SRVLCNT

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).

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'244C'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 16
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SRVLST* on page 725

NAME

SRVLSRV — Server List Entry

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'244D'

Length *

Class CLASS

Sprcls SCALAR

Server List Entries (SRVLSRV) contains information for one target server in the server list SRVLST. This information consists of a weighted priority used for workload balancing and either an SNA address or a TCPIP address associated with the target server. The choice of TCPIP or SNA 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 an IPADDR. For an SNA connection, a server list entry consists of an SRVPRTY and an SNAADDR.

A server list consists of a list of (SRVPRTY, IPADDR) or (SRVPRTY, SNAADDR) pairs.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'244D'	
srvprty	INSTANCE_OF REQUIRED NOTE REPEATABLE	SVRPRTY Required with SNAADR or IPADDR
snaaddr	INSTANCE_OF MTLEXC REPEATABLE	SNAADDR - SNA Address X'11E8' - IPADDR - TCP/IP Address
ipaddr	INSTANCE_OF MTLEXC REPEATABLE	IPADDR - TCP/IP Address X'11E9' - SNAADDR - SNA Address
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SRVLST* on page 725

NAME

SRVLST — Server List

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'244E'

Length *

Class CLASS

Sprcls 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.

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.

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.

Caching of 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'244E'	
srvlcnt	INSTANCE_OF REQUIRED	SRVLCNT - Server List Count
srvlsrv	INSTANCE_OF REPEATABLE REQUIRED	SRVLSRV - Server List Entry
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDBRM* on page 48
semantic	*ACCRDBRM* on page 48
	SRVLCNT on page 722
	SRVLSRV on page 723
	SRVPRTY on page 730

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'116D'	
value	INSTANCE_OF REQUIRED	NAMDR - Name Data
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
	EXCSATRD on page 329
semantic	*EXCSAT* on page 323
vldattls	*SERVER* on page 670

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 3-75. 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 server handles both files and relational databases. Multiple servers can reside in the same host system.

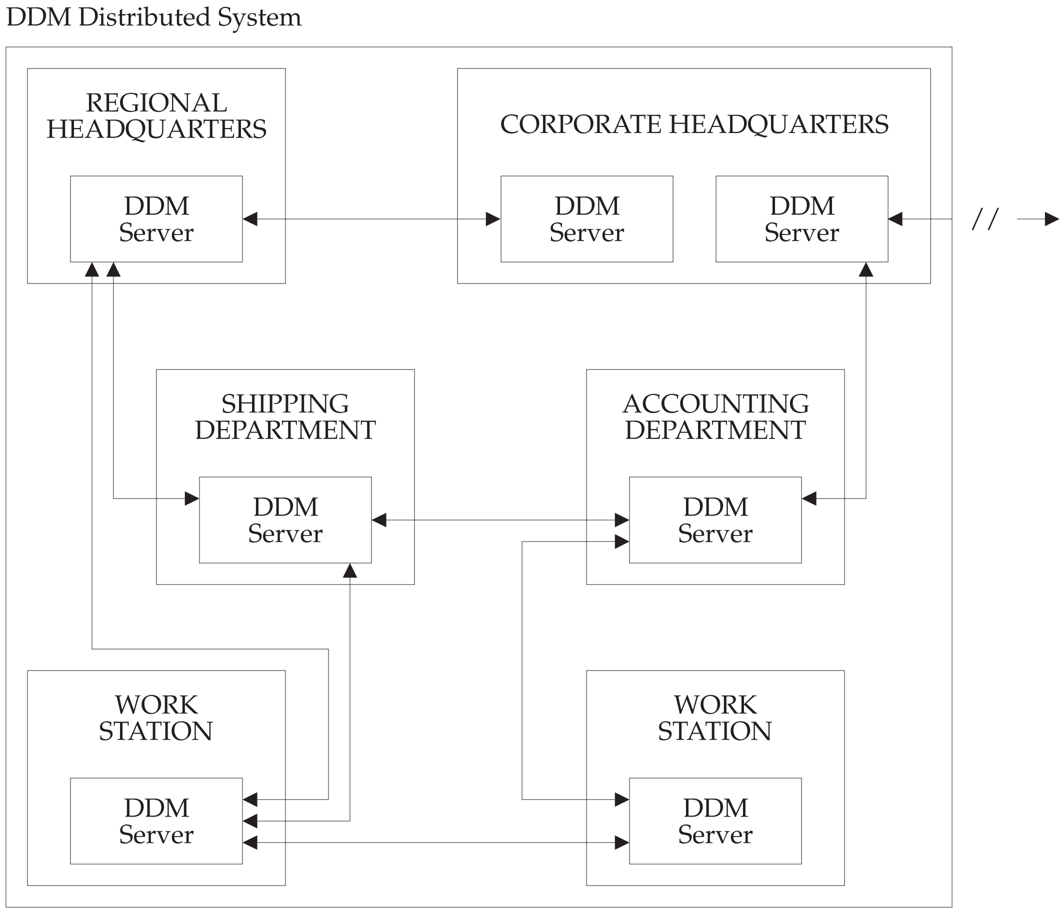


Figure 3-75 DDM Distributed Systems and Servers

SEE ALSO

Variable	Reference
semantic	*DDM* on page 255

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 the term SRVLST for a description of workload balancing.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'244F'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 16
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*SRVLST* on page 725

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'115A'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*EXCSAT* on page 323
	EXCSATRD on page 329
semantic	*EXCSAT* on page 323

STGLMT

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

Variable	Reference
insvar	*RSCTYP* on page 638
semantic	*RSCTYP* on page 638

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

Variable	Reference
insvar	*STTSTRDEL* on page 744
semantic	*ENDBND* on page 307
	STTSTRDEL on page 744

STRDELDQ

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

Variable	Reference
insvar	*STTSTRDEL* on page 744
semantic	*STTSTRDEL* on page 744

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0038'	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
semantic	*ARRAY* on page 96
	INHERITANCE on page 380
sprcls	*BNDCHKEXS* on page 119
	BNDCRTCTL on page 121
	BNDEXPOPT on page 123
	BNDOPTNM on page 128
	BNDOPTVL on page 129
	BNDSTTASM on page 135
	CCSIDDBC on page 138
	CCSIDMBC on page 139
	CCSIDSBC on page 144
	CRRTKN on page 240
	DEPERRCD on page 265
	DSCERRCD on page 277
	ELMCLS on page 306
	ENUCLS on page 316
	FDODSC on page 354
	INSTANCE_OF on page 387

Variable	Reference
	LOGNAME on page 410
	LOGTSTMP on page 411
	NAME on page 445
	NOTE on page 452
	PKGATHOPT on page 493
	PKGDFTCST on page 503
	PKGISOLVL on page 505
	PKGRPLOPT on page 516
	PKGSNLST on page 520
	PRCCNVCD on page 521
	PRDDTA on page 528
	PRDID on page 529
	QRYBLKCTL on page 557
	QRYPRCTYP on page 566
	RDBACCCL on page 577
	RDBRLSOPT on page 603
	RSCNAM on page 637
	RSCTYP on page 638
	RSNCOD on page 643
	SECCHKCD on page 656
	SECMEC on page 661
	SPRCLS on page 678
	SRVDGN on page 720
	SRVRLSLV on page 731
	STTDATFMT on page 741
	STTDECDEL on page 742
	STTSTRDEL on page 744
	STTTIMFMT on page 746
	SVCERRNO on page 755
	SYNERRCD on page 831
	TEXT on page 864

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 the term OOPOVR 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 3-76 on page 738. The DDM architecture, like nature, consists of layered objects, as illustrated in Figure 3-77 on page 739. 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.

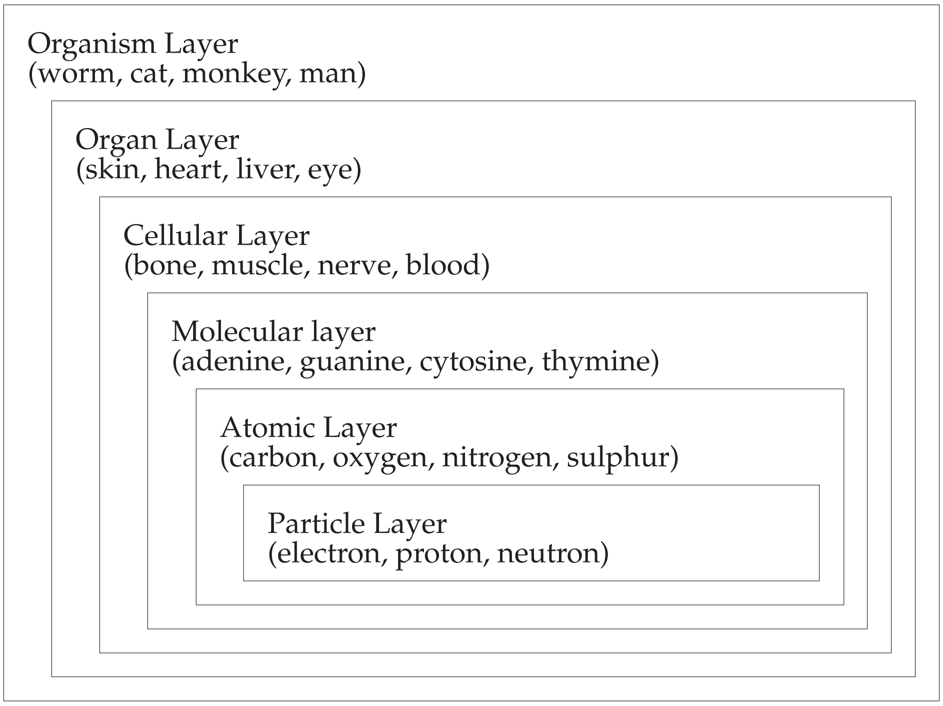


Figure 3-76 Structural Layers in Nature

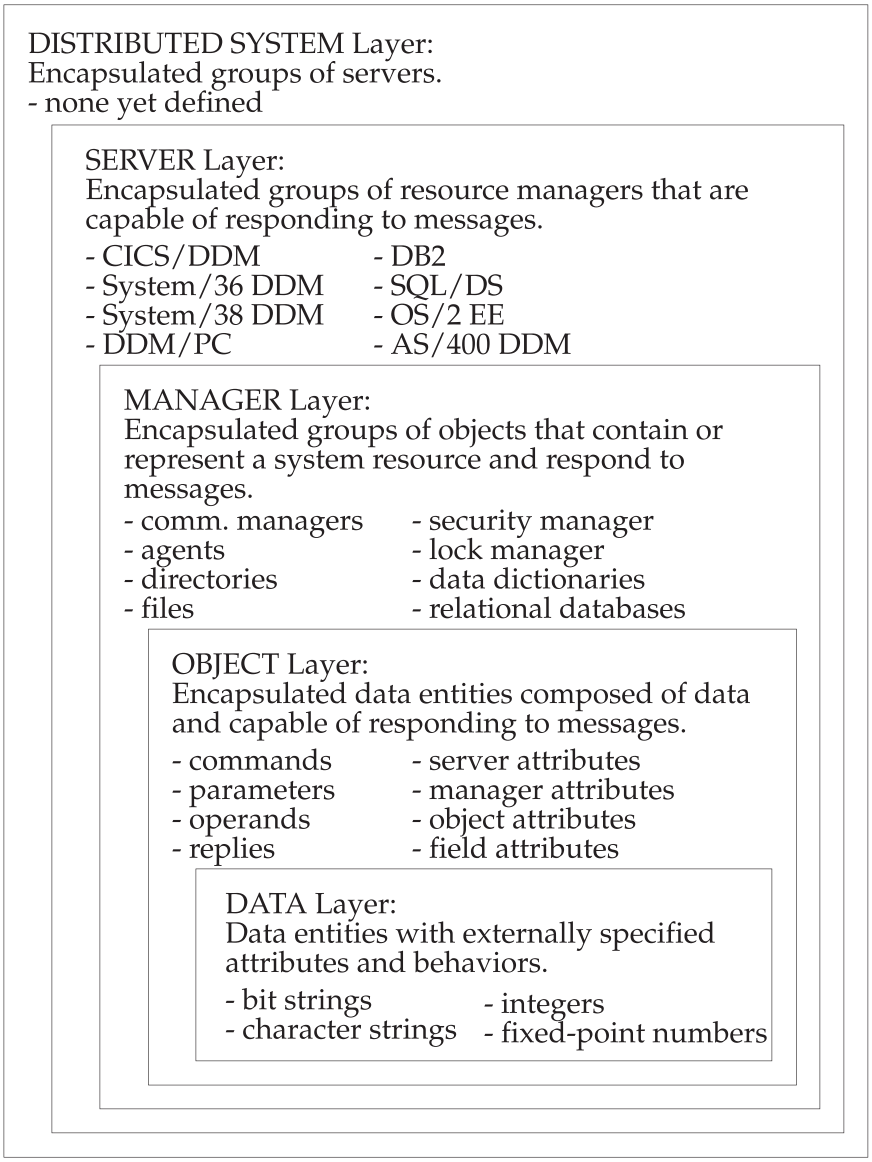


Figure 3-77 The Structural Layers of DDM Architecture

SEE ALSO

Variable	Reference
semantic	*CONCEPTS* on page 237

STTASMEUI

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

Variable	Reference
insvar	*BNDSTTASM* on page 135

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 - Code Point 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
	DFTVAL	X'2429' - ISODATFMT - ISO Date Format
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606
	STTTIMFMT on page 746

STTDECDEL

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2121'	
value	INSTANCE_OF	CODPNTDR - Code Point 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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
	BGNBND on page 101
semantic	*ACCRDB* on page 42
	BGNBND on page 101
	DFTPKG on page 267
	ENDBND on page 307
	REBIND on page 606

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

Variable	Reference
insvar	*BNDSTTASM* on page 135

STTSTRDEL

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 STRDELDQ, then the string delimiter in SQL statements is a quote character, and the delimiter for SQL statements is an apostrophe character.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2120'	
value	INSTANCE_OF	CODPNTDR - Code Point Data Representation
	ENUVAL	X'2426' - STRDELAP - String Delimiter Apostrophe
	ENUVAL	X'2427' - STRDELDQ - 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.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
	BGNBND on page 101
semantic	*ACCRDB* on page 42

Variable	Reference
	BGNBND on page 101
	DFTPKG on page 267
	ENDBND on page 307
	REBIND on page 606

STTTIMFMT

NAME

STTTIMFMT — Statement Time Format

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'2123'

Length *

Class CLASS

Sprcls STRING - String

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'2123'	
value	INSTANCE_OF	CODPNTDR - Code Point 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
	DFTVAL	X'242E' - ISOTIMFMT - ISO Time Format
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
semantic	*BGNBND* on page 101
	ENDBND on page 307
	REBIND on page 606
	STTDATFMT on page 741

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 the description of the term *SERVER* on page 670. 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:[35]
  - *Supervisor*—See the description of the term *SUPERVISOR* on page 753.
  - *Security Manager*—The level of security DDM defines is required of all target servers. See the description of the term *SECMGR* on page 663.
  - *Dictionary*—The tables of descriptors for code point identified descriptors of transmitted DDM terms and product extension terms. See the description of the term *DICTIONARY* on page 272. 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 *CMNAPPC* on page 179)
   - **CMNSYNCPT—**SNA LU 6.2 Sync Point Conversational Communications Manager (see *CMNSYNCPT* on page 197)
   - **CMNTCPIP—**TCP/IP Communication Manager (see *CMNTCPIP* on page 209)

   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 the term *AGENT* on page 56 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.

---

35. 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 694)

     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 782)

     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.

   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 code points in the dictionaries QDDPRMD, QDDBASD, and QDDRDBD. While product-pair agreements to support product extensions are permitted, they cannot be required.
2. Product extension code points 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:

TRGNSPRM
: Target Not Supported

CMDNSPRM
: Command Not Supported

PRMNSPRM
Parameter Not Supported

VALNSPRM
Parameter Value Not Supported

OBJNSPRM
Object Not Supported

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 code points of valid DDM structures. These are:

1. The basis for recognizing a command is the class of object to which it is sent (see the figure in the description of the term *AGNCMDPR* on page 60). 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 the descriptions of the terms *DSS* on page 289 and *RQSCRR* on page 626). 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 the descriptions of the terms *DSS* on page 289 and *RQSCRR* on page 626). 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 the descriptions of the terms *DSS* on page 289 and *RQSCRR* on page 626). 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

Variable	Reference
clsvar	*MANAGER* on page 417
semantic	*CMDNSPRM* on page 173
	CONCEPTS on page 237
	EXTENSIONS on page 346
	INHERITANCE on page 380
	OPTIONAL on page 485
	PRCOVR on page 526
	PRMNSPRM on page 531
	REQUIRED on page 615
	VALNSPRM on page 898

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 3-20 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 the EXCSAT command.

Table 3-20 Supervisor-Level Compatibility

DDM Levels	**1**	**2**	**3**	**4**	**5**
SUPERVISOR	1	1	1	1	1
Commands					
EXCSAT	1	1	1	1	1

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'143C'	
secmgr	INSTANCE_OF	SECMGR - Security Manager
clscmd	**NIL**	
inscmd		**INSTANCE COMMANDS**
1041	INSTANCE_OF REQUIRED	EXCSAT - Exchange Server Attributes
mgrlvln	1	

mgrdepls	**NIL**	
vldattls		**VALID ATTRIBUTES**
0019	INSTANCE_OF	HELP - Help Text
115D	INSTANCE_OF	SPVNAM - Supervisor Name
0045	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
insvar	*DEPERRCD* on page 265
	MGRLVL on page 427
semantic	*ACCRDB* on page 42
	AGENT on page 56
	EXTENSIONS on page 346
	INHERITANCE on page 380
	MANAGER on page 417
	RDBOVR on page 593
	SUBSETS on page 747

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11B4'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 32
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SECCHKRM* on page 659

SVRCOD

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	6	
class	X'1149'	
value	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	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ABNUOWRM* on page 39
	ACCRDBRM on page 48
	AGNPRMRM on page 61
	BGNBNDRM on page 109
	CMDATHRM on page 168
	CMDCHKRM on page 170
	CMDCMPRM on page 172
	CMDNSPRM on page 173
	CMDVLTRM on page 176

Variable	Reference
	CMMRQSRM on page 177
	DSCINVRM on page 279
	DTAMCHRM on page 303
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	MGRDEPRM on page 426
	MGRLVLRM on page 432
	OBJNSPRM on page 462
	OPNQFLRM on page 474
	OPNQRYRM on page 483
	PKGBNARM on page 497
	PKGBPARM on page 498
	PRCCNVRM on page 523
	PRMNSPRM on page 531
	QRYNOPRM on page 562
	QRYPOPRM on page 564
	RDBACCRM on page 578
	RDBAFLRM on page 579
	RDBATHRM on page 581
	RDBNACRM on page 587
	RDBNFNRM on page 592
	RDBUPDRM on page 604
	RPYMSG on page 624
	RSCLMTRM on page 634
	RSLSETRM on page 642
	SECCHKRM on page 659
	SQLERRRM on page 710
	SYNERRCD on page 831
	SYNTAXRM on page 835
	TRGNSPRM on page 868
	VALNSPRM on page 898
semantic	*AGNRPYPR* on page 63
	APPTRGER on page 91
	DCESECOVR on page 248
	LMTBLKPRC on page 400

Variable	Reference
	SECCHK on page 652
	SECCHKCD on page 656
	SECCHKRM on page 659
	TCPTRGER on page 861
	USRSECOVR on page 893

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.

DSS Carrier: OBJDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'1248'	
synctype	INSTANCE_OF NOTE REQUIRED	SYNCTYPE - Sync point operation type The only valid values of synctype are X'03' Unit of work committed X'04' Unit of work rolled back X'05' Request to commit unit of work X'06' Forget unit of work X'0A' Migrated unit of work

SEE ALSO

Variable	Reference
rpydta	*SYNCCTL* on page 760
semantic	*SYNCCTL* on page 760
	SYNCPTOV on page 787

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.

Source System Processing

The SYNCCTL command is used 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

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.

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.

DSS Carrier: RQSDSS

Target System Processing

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 SYNCCRD 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 SYNCPTMGR state table in SYNCPTOV for information.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1055'	
synctype	INSTANCE_OF REQUIRED CMDTRG	SYNCTYPE - Sync point operation type
rlsconv	INSTANCE_OF OPTIONAL NOTE	RLSCONV - Release Conversation This parameter can only be specified if synctype is: X'01' (prepare to commit) X'05' (request to commit) X'06' (forget and terminate a conversation) Otherwise this parameter must not be present.
uowid	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier This parameter must be specified if synctype is: X'06' (forget unit of work) X'09' (new unit of work) Otherwise this parameter must not be present.

forget	INSTANCE_OF LENGTH NOTE ENUVAL NOTE ENUVAL NOTE DFTVAL	FORGET - Forget Unit of Work 2 This parameter must be specified if synctype is: X'03' (unit of work committed) X'03' (forget) Otherwise this parameter must not be present. X'F1' - TRUE - True State X'F1' - Forget reply required. Required if release was set on SYNCCTL synctype(prepare). X'F0' - FALSE - False State X'F0' - Next reply implies forget. X'F1' - TRUE - True State
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'106F'	INSTANCE_OF OPTIONAL REPEATABLE NOTE	SYNCLOG - Sync Point Log 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.
rpydta		**REPLY OBJECTS**
X'1248'	INSTANCE_OF OPTIONAL NOTE	SYNCCRD - Sync Point Control Reply SYNCCRD is sent in response to a SYNCCTL command if the synctype is: X'01' (prepare to commit) X'02' (migrate unit of work) X'03' (committed and forget set to TRUE) X'05' (request to commit)
X'106F'	INSTANCE_OF OPTIONAL NOTE	SYNCLOG - Sync Point Log SYNCLOG is returned only when synctype is set to X'08' (request log).
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
insvar	*PRCCNVCD* on page 521
semantic	*SYNCPTOV* on page 787

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'106F'	
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
logname	INSTANCE_OF REQUIRED	LOGNAME - Log Name
logtstmp	INSTANCE_OF REQUIRED	LOGTSTMP - Log Timestamp
cnntkn	INSTANCE_OF REQUIRED	CNNTKN - Connection Token
snaaddr	INSTANCE_OF NOTE MTLEXC	SNAADDR - SNA Address resync address for SNA conversations. X'11E8' - IPADDR - TCP/IP Address
ipaddr	INSTANCE_OF NOTE MTLEXC	IPADDR - TCP/IP Address resync address for TCP/IP conversations. X'11E9' - SNAADDR - SNA Address
tcphost	INSTANCE_OF OPTIONAL NOTE MTLEXC	TCPHOST - TCP/IP Domain Qualified Host Name TCP/IP host name of CMNTCPIP. X'11E9' - SNAADDR - SNA Address
clscmd	**NIL**	

inscmd	**NIL**

SEE ALSO

Variable	Reference
cmddta	*SYNCCTL* on page 760
	SYNCRSY on page 828
rpydta	*SYNCCTL* on page 760
semantic	*SYNCCTL* on page 760
	SYNCPTOV on page 787
	SYNCRSY on page 828

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 3-78. 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 the term SYNCMNI 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.

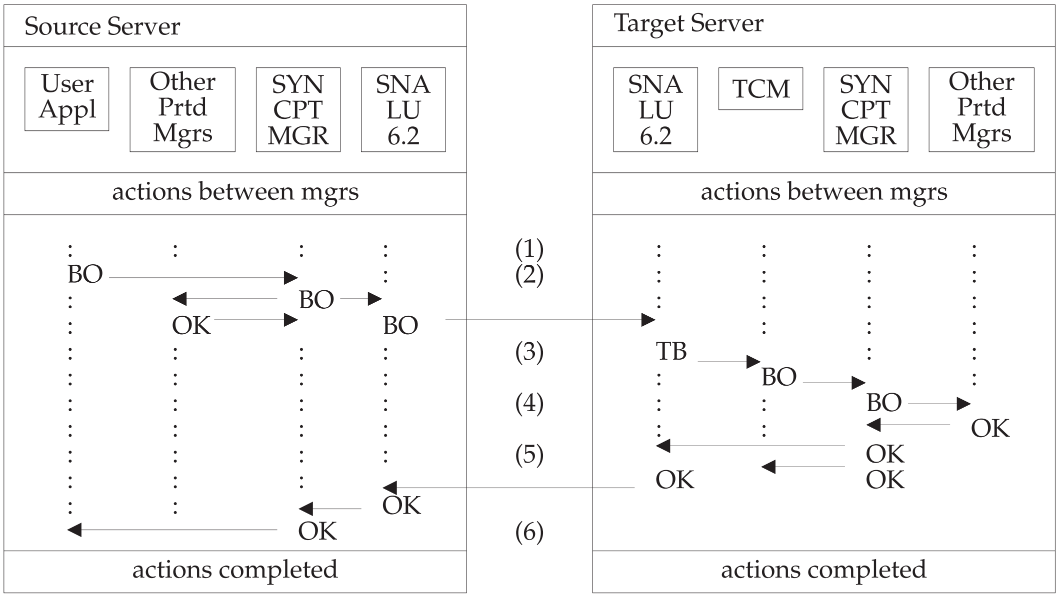


Figure 3-78 Backout Flows Between Managers (SYNCMNBK) Protocol

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

Variable	Reference
semantic	*CMNSYNCPT* on page 197

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 3-79. 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 the term SYNCMNI with ALLOCATE(SYNC_LEVEL(SYNCPT)).
- No error situation occurs.

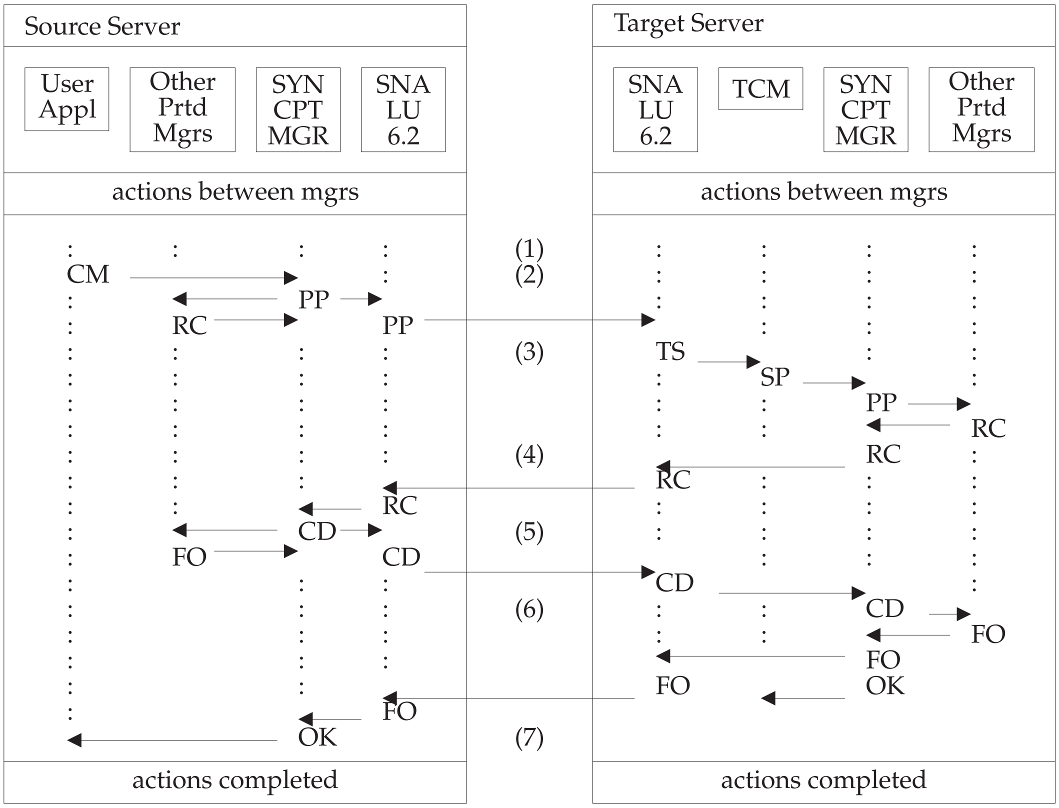


Figure 3-79 Two-Phase Commit Flows Between Managers (SYNCMNCM) Protocol

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

Variable	Reference
semantic	*CMNSYNCPT* on page 197
	SYNCMNT on page 778

NAME

SYNCMNFL — LU 6.2 Sync Point Communications Failure

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length *

Class HELP

LU 6.2 Sync Point Communications Failure (SYNCMNFL) 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 3-80 on page 772 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 the term SYNCMNI.)
- 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 the term RESYNOVR and the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM).

Source Communications Mgr.		Target Communications Mgr.
send state		receive state
SEND_DATA RESOURCE(res_var) DATA(cmd_var) LENGTH(lng_var) RETURN_CODE(rc_var)	(1)	RECEIVE_AND_WAIT RESOURCE(res_var) FILL(LL) LENGTH(lng_var) RETURN_CODE(rc_var)
REQUEST_TO_SEND_RECEIVED (rts_var) : :	FAILURE (2)	REQUEST_TO_SEND_RECEIVED (rts_var) DATA(dta_var) WHAT_RECEIVED(typ_var) :
rc_var = RESOURCE_FAILURE_ENTRY rts_var = NO	(3) (4)	lng_var = (unchanged) rts_var = NO rc_var = RESOURCE_FAILURE_RETRY typ_var = (not set)
backout_required state	(5)	backout_required state
BACKOUT	(6)	BACKOUT
(cleanup)	(7)	(cleanup)
DEALLOCATE RESOURCE(res_var) TYPE(LOCAL) RETURN_CODE(rc_var) : rc_var = OK	(8)	DEALLOCATE RESOURCE(res_var) TYPE(LOCAL) RETURN_CODE(rc_var) : rc_var = OK

Figure 3-80 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 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

Variable	Reference
semantic	*CMNSYNCPT* on page 197
	SYNCMNT on page 778

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 3-81 on page 775. 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.

Source Communications Mgr.		Target Communications Mgr.
ALLOCATE LU_NAME(OTHER(target)) MODE_NAME(mode) TPN(tp_name) TYPE(BASIC_CONVERSATION) RETURN_CONTROL (WHEN_SESSION_ALLOC.) SYNC_LEVEL(SYNCPT) SECURITY(PGM(USER_ID(xx) PASSWORD(yyy))) PIP(NO) RESOURCE(res_var) RETURN_CODE(rc_var) : :	(1)	
:	(2) →	session established
res_var = xxxxx rc_var = OK	(3)	
SEND_DATA RESOURCE(res_var) DATA(cmd_var) LENGTH(lng_var) RETURN_CODE(rc_var) REQUEST_TO_SEND_RECEIVED (rts_var) :	(4)	
rts_var = NO rc_var = OK	(5)	
RECEIVE_AND_WAIT	(6) →	ATTACH processed comm mgr started RECEIVE_AND_WAIT . . .

Figure 3-81 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

Variable	Reference
semantic	*APPSRCCD* on page 75
	APPSRCCR on page 82
	APPSRCER on page 87
	APPTRGER on page 91
	CMNSYNCPT on page 197
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCMNT on page 778

SYNCMNT

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 the term SYNCMNFL 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 3-82 on page 779. 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 the term SYNCMNI 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.

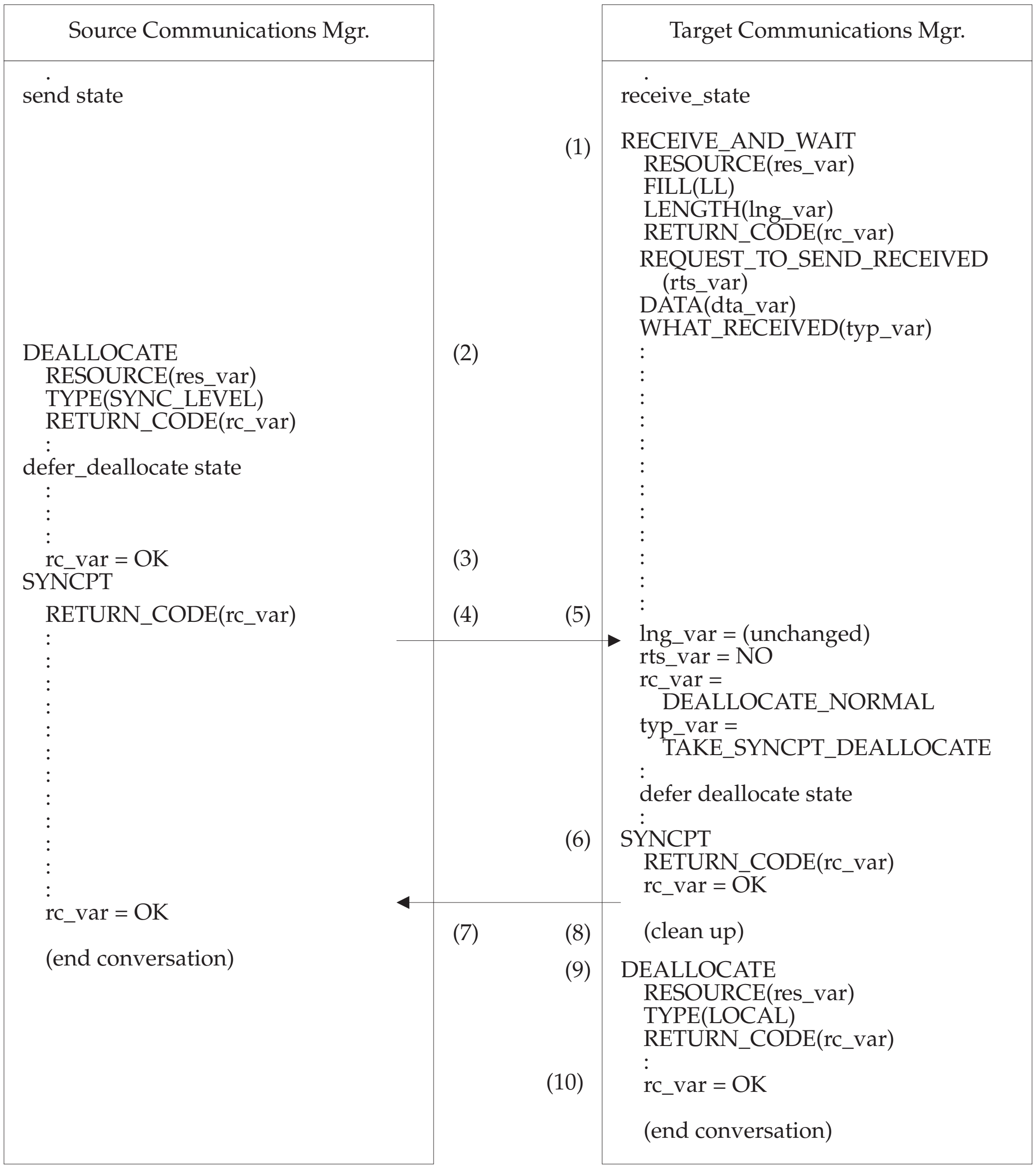


Figure 3-82 Normal Communications Termination (SYNCMNT) Protocol

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 the term SYNCMNCM 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
- A 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

Variable	Reference
semantic	*CMNSYNCPT* on page 197

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 3-83 on page 783.

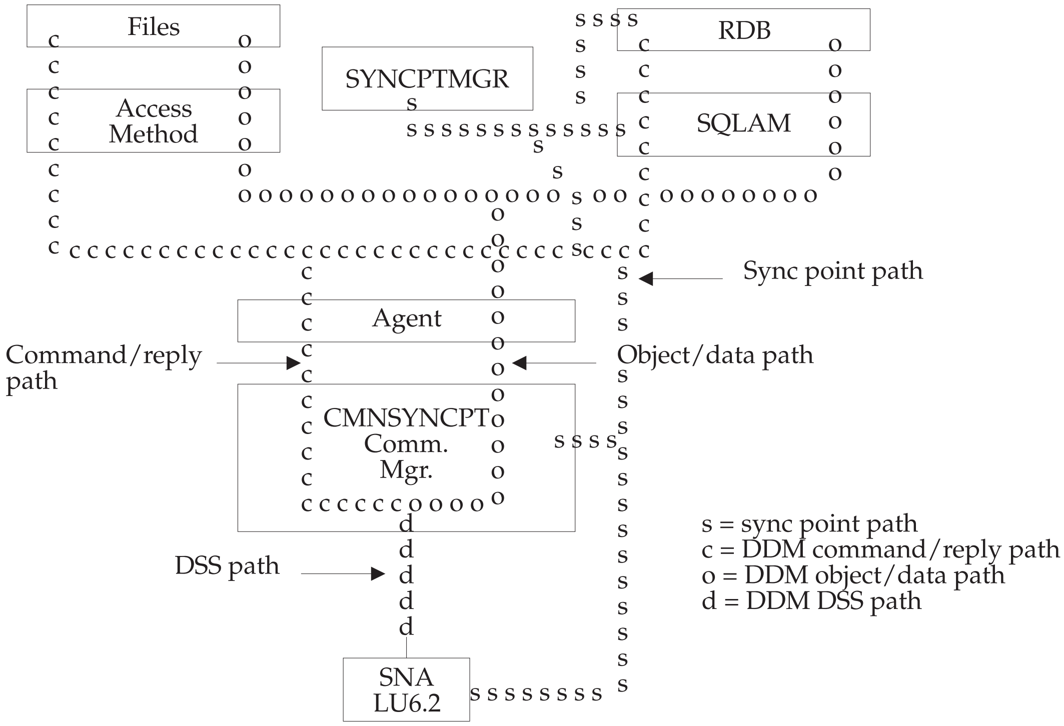


Figure 3-83 Server Paths for SYNCPTMGR at DDM Level 4

The role of the SYNCPTMGR at DDM Level 5 is illustrated in Figure 3-84 on page 784. 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.

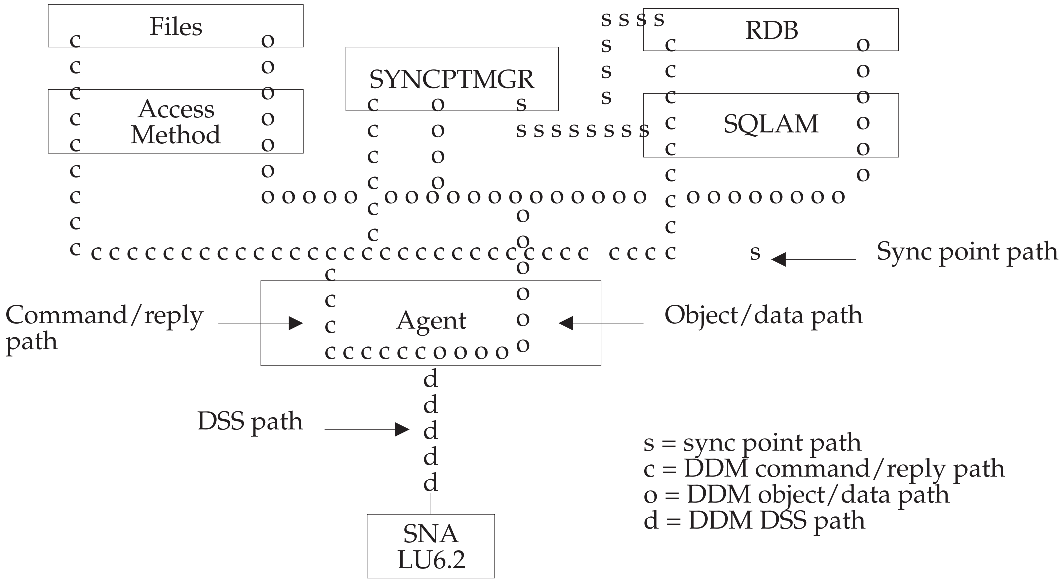


Figure 3-84 Server Paths for SYNCPTMGR at DDM Level 5

Manager-Level Compatibility

Table 3-21 illustrates the function of the SYNCPTMGR as it has grown and changed through the levels of DDM architecture.

Table 3-21 Sync Point Manager-Level Compatibility

DDM Levels	1	2	3	4	5
SYNCPTMGR				4	5
Manager Dependencies					
CMNSYNCPT				4	
RSYNCMGR (required for resync server support)					5
CMNAPPC					3
CMNTCPIP					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 help term SYNCPTOV.

clsvar	NIL
insvar	NIL
clscmd	NIL
inscmd	NIL
mgrlvln	4

mgrdepls		**MANAGER DEPENDENCY LIST**
X'147C'	INSTANCE_OF	CMNSYNCPT - SNA LU 6.2 Sync Point Conversational Communications Manager
	MGRLVLN	4
	NOTE	The communications manager must support SNA LU 6.2 SYNC_LEVEL(SYNCPT).
X'14C1'	INSTANCE_OF	RSYNCMGR - Resynchronization Manager
	MGRLVLN	5
	NOTE	Required only for resync server support.
X'1444'	INSTANCE_OF	CMNAPPC - LU 6.2 Conversational Communications Manager
	MGRLVLN	3
	NOTE	The communications manager must support SNA LU 6.2 SYNC_LEVEL(NONE).
X'1474'	INSTANCE_OF	CMNTCPIP - TCP/IP Communication Manager
	MGRLVLN	5
X'1403'	INSTANCE_OF	AGENT - Agent
	MGRLVLN	5
vldattls		**VALID ATTRIBUTES**
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

SEE ALSO

Variable	Reference
cmddta	*SYNCRSY* on page 828
insvar	*MGRLVL* on page 427
mgrdepls	*CMNSYNCPT* on page 197
semantic	*CMNSYNCPT* on page 197
	INHERITANCE on page 380
	LOGNAME on page 410
	LOGTSTMP on page 411
	MANAGER on page 417
	RDBOVR on page 593
	RESYNOVR on page 618
	SQLAM on page 694
	SUBSETS on page 747

Variable	Reference
	SYNCCRD on page 759
	SYNCCTL on page 760
	SYNCLOG on page 764
	SYNCMNBK on page 766
	SYNCMNCM on page 768
	SYNCMNFL on page 771
	SYNCPTOV on page 787
	SYNCRRD on page 826
	SYNCRSY on page 828
	SYNCTYPE on page 830
	UOWSTATE on page 885
title	*SYNCPTOV* on page 787

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.

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 3-85 on page 789 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.

Source Server		Target Server
open (conversation)		accept (conversation)
EXCSAT (Exchange Server Attributes)	→	
	←	EXCSATRD (Exchange Server reply)
ACCSEC (Access Security)	→	
	←	ACCSECRD (ACCSEC reply)
SECCHK (Security Check)	→	
	←	SECCHKRD (SECCHK reply)
ACCRDB (Access RDB)	→	
	←	ACCRDBRM (ACCRDB reply)
SYNCCTL (Request Log Information)	→	
	←	SYNCLOG (Target Log Information) (Target Connection Token)
SYNCCTL (New UOW ID)	→	
	←	Begin UOW
EXCSQLIMM (Execute Immediate)	→	
	←	RDBUPDRM (RDB update) SQLCARD (SQLCA)
SYNCCTL (Prepare, Release)	→	
SYNCLOG (Source Log Information) (Source Connection Token)	→	
	←	SYNCCRD (Request Commit)
SYNCCTL (Committed, Forget)	→	
	←	SYNCCRD (Forget)
close (conversation)		close (conversation)

Figure 3-85 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.

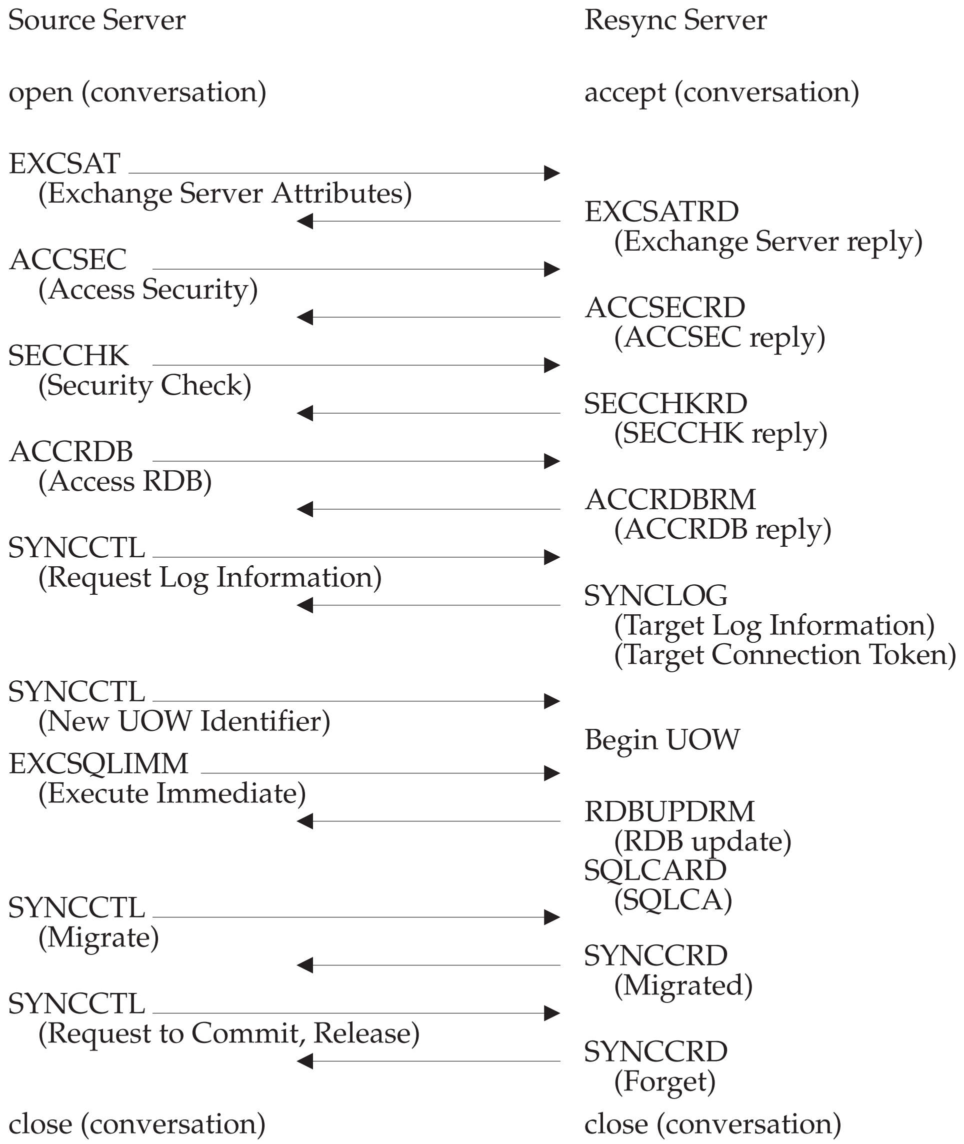


Figure 3-86 Illustration of Resync Server Sync Point Flow

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 rollbacked 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.

Source Server　　Target Server 1　　Target Server 2

SYNCCTL
  synctype(prepare)
  release(TRUE)
SYNCLOG
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(source rdbnam)
  cnntkn(source cnntkn)

　　SYNCCRD
　　 synctype(request to commit)

SYNCCTL
  synctype(prepare)
  release(TRUE)
SYNCLOG
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(source rdbname)
  cnntkn(source cnntkn)

　　　　SYNCCRD
　　　　 synctype(request to commit)

SYNCCTL
  synctype(committed, forget)

　　SYNCCRD
　　 synctype(forget)

close (conversation)　　close (conversation)

SYNCCTL
  synctype(committed, forget)

　　　　SYNCCRD
　　　　 synctype(forget)

close (conversation)　　　　close (conversation)

Figure 3-87 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 SYNCCRD forget reply in this example because the connections are being terminated at part of the sync point operation.

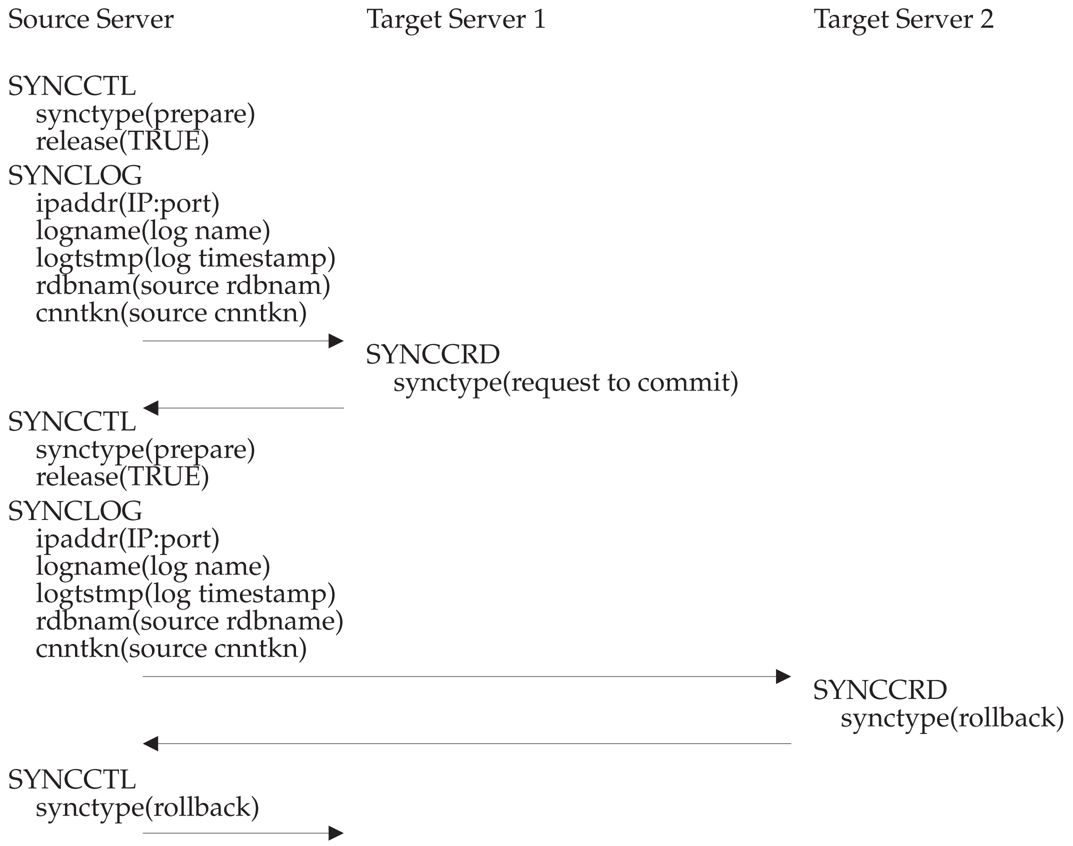


Figure 3-88 Multiple Released Connections Rollback Flow

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.

Source Server | Target Server

SYNCCTL
synctype(prepare)
release(FALSE)
SYNCLOG
ipaddr(IP:port)
logname(log name)
logtstmp(log timestamp)
rdbnam(source rdbname)
cnntkn(source cnntkn)

SYNCCRD
synctype(request to commit)

SYNCCTL
synctype(committed)
forget(TRUE)

SYNCCRD
synctype(forget)

Figure 3-89 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.

Source Server | Target Server

SYNCCTL
synctype(prepare)
release(FALSE)
SYNCLOG
ipaddr(IP:port)
logname(log name)
logtstmp(log timestamp)
rdbnam(source rdbname)
cnntkn(source cnntkn)

SYNCCRD
synctype(forget)

Figure 3-90 Commit Without Updates Flow

The above flow illustrates the two sync point control messages exchanged when the server is read-only.

Source Server	Target Server
SYNCCTL synctype(prepare) release(TRUE) SYNCLOG ipaddr(IP:port) logname(log name) logtstmp(log timestamp) rdbnam(source rdbname) cnntkn(source cnntkn) →	
	SYNCCRD synctype(request commit)
← SYNCCTL synctype(committed) forget(TRUE) →	
	SYNCCRD synctype(forget)
←	
close(conversation)	close(conversation)

Figure 3-91 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.

Source Server	Target Server
SYNCCTL synctype(rollback) release(no) →	
	Rollback unit of work
	No reply

Figure 3-92 Source SYNCPTMGR Initiated Rollback

The above flow illustrates the one sync point control message when a source server with a log initiates a rollback.

```
Source Server                          Target Server

SYNCCTL
  synctype(rollback)
  release(FALSE)
SYNCLOG
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(source rdbname)
  cnntkn(source cnntkn)
         ---------------------------->
                                       SYNCCRD
                                         synctype(rollback)
         <----------------------------
```

Figure 3-93 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.

```
Source Server                          Target Server

SYNCCTL
  synctype(forget)
  forget(TRUE)
  release(TRUE)
         ---------------------------->
                                       SYNCCRD
                                         synctype(forget)
         <----------------------------
close(conversation)                    close(conversation)
```

Figure 3-94 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.

```
Source Server                          Target Server

SYNCCTL
  synctype(prepare)
  release(yes)
SYNCLOG
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(source rdbname)
  cnntkn(source cnntkn)
        ------------------------->
                                       SYNCCRD
                                         synctype(rollback)
        <-------------------------
```

Figure 3-95 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 3-96 on page 800).

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 3-94 on page 798).

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:

- 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

```
Source Server                          Target Server

SYNCCTL
  synctype(prepare)
  release(FALSE)
SYNCLOG
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(source rdbname)
  cnntkn(source cnntkn)
         --------------------------->
                                       SYNCCRD
                                         synctype(request commit)
         <---------------------------
SYNCCTL
  synctype(committed)
  forget(FALSE)
         --------------------------->
Next DDM request
         --------------------------->
                                       Next DDM reply
         <---------------------------
Process reply as a forget
(implied forget)
```

Figure 3-96 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 SYNCCRD committed reply to the coordinator, and commits. The coordinator is in the committing state. The coordinator sends each of the other participants the SYNCCTL committed command.

- If all participants voted forget", the coordinator sends the SYNCCTL 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 SYNCCRD forget reply message to the coordinator to complete the unit of work.
- If a participant voted rollback, the coordinator sends the SYNCCTL 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 SYNCCTL rollback command to all participants who did not vote rollback or forget. The resync server is not sent a SYNCCTL 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 SYNCCTL 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 SYNCCTL migrate and SYNCCTL request to commit commands. If there are multiple target servers this chaining is not allowed.

```
Source Server                          Target Server
                                       (Resync Server)

SYNCCTL
  synctype(migrate)
SYNCCTL
  synctype(request to commit)
          ---------------------------> 
                                       SYNCCRD
                                         synctype(migrated)
                                       SYNCCRD
                                         synctype(forget)
          <---------------------------
```

Figure 3-97 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 3-98 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.

```
Source Server              Target Server 1                 Target Server 2

SYNCCTL
  synctype(migrate)
          ---------------------------------------------->
                                                           SYNCCRD
                                                             synctype(migrated)
          <----------------------------------------------
SYNCCTL
  synctype(prepare)
  release(TRUE)
SYNCLOG
  (SYNCLOG for server 2)

  ipaddr(Server 2 IP:port)
  logname(Server 2 logname)
  lgtstmp(Server 2 timestamp)
  rdbnam(Server 2 rdbname)
  cnntkn(Server 2 cnntkn)
          --------------->
                            SYNCCRD
                              synctype(request to commit)
          <---------------
SYNCCTL
  synctype(request to commit)
  release(TRUE)
SYNCLOG
  (SYNCLOG for server 1)

  ipaddr(Server 1 IP:port)
  logname(Server 1 logname)
  lgtstmp(Server 1 timestamp)
  rdbnam(Server 1 rdbname)
  cnntkn(Server 1 cnntkn)
          ---------------------------------------------->
                                                           SYNCCRD
                                                             synctype(committed)
          <----------------------------------------------
SYNCCTL
  synctype(committed)
  forget(TRUE)
          --------------->
                           SYNCCRD
                             synctype(forget)
          <---------------

                           close (conversation)
SYNCCTL
  synctype(forget)
  uowid(UOW ID)
          ---------------------------------------------->

close (conversation)                                       close (conversation)
```

Figure 3-99 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.

Source Server | Target Server 1 | Target Server 2 (Resync Server)

SYNCCTL
synctype(migrate)
→ SYNCCRD
synctype(migrated)
←
SYNCCTL
synctype(prepare)
release(TRUE)
SYNCLOG
(SYNCLOG for server 2)
ipaddr(Server 2 IP:port)
logname(Server 2 logname)
lgtstmp(Server 2 timestamp)
rdbnam(Server 2 rdbname)
cnntkn(Server 2 cnntkn)
→ SYNCCRD
synctype(request to commit)
←
SYNCCTL
synctype(request to commit)
release(TRUE)
SYNCLOG
(SYNCLOG for server 1)
ipaddr(Server 1 IP:port)
logname(Server 1 logname)
lgtstmp(Server 1 timestamp)
rdbnam(Server 1 rdbname)
cnntkn(Server 1 cnntkn)
→ SYNCCRD
synctype(rollback)
←
SYNCCTL
synctype(rollback)
→

Figure 3-100 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.

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 SYNCPTMGR. 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 SYNCPTMGR 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 rollbacked or forgotten. (Specified on the resync sync point command or reply by the coordinator or resync server when the coordinator has rollbacked 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 SYNCPTMGRs 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 SYNCLOG 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 SYNCLOG 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 SYNCPTMGRs.

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 rollbacked.

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 3-101 on page 808 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.

Source Server
(participant)

Target Server
(coordinator)

EXCSAT
(Exchange Server Attributes)

extname(external name)
mgrlvls(manager level list)
srvclsnm(server class name)
srvnam(server name)
srvrlsvl(server release level)

EXCSATRD
(EXCSAT reply data)

extnam(external name)
mgrlvls(manager level list)
srvclsnm(server class name)
srvnam(server name)
srvrlsvl(release level)

SYNCRSY
(Sync Point Resync)
rsynctyp(UOW)
uowid(unit of work identifier)
uowstate(in-doubt)

SYNCLOG
(Participant Log SYNCLOG)

ipaddr(IP:port)
logname(log name)
logtstmp(log timestamp)
rdbname(rdbname)
cnntkn(cnntkn)

SYNCLOG
(Coordinator Log SYNCLOG)

ipaddr(IP:port)
logname(log name)
logtstmp(log timestamp)
rdbname(rdbname)
cnntkn(cnntkn)

SYNCRRD
(SYNCRSY Reply Data)
rsynctyp(UOW)
uowstate(commit)
uowid(unit of work ID)

Figure 3-101 Resync Connection (Part 1)

```
Source Server                              Target Server
(participant)                              (coordinator)

SYNCRSY
  (Sync Point Resync)
  rsynctyp(UOW)
  uowid(unit of work identifier)
  uowstate(in-doubt)

SYNCLOG
  (Participant Log SYNCLOG)
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(rdbname)
  cnntkn(cnntkn)

SYNCLOG
  (Coordinator Log SYNCLOG)
  ipaddr(IP:port)
  logname(log name)
  logtstmp(log timestamp)
  rdbnam(rdbname)
  cnntkn(cnntkn)
         ------------------------------>
                                           SYNCRRD
                                             (SYNCRSY Reply Data)
                                             rsynctyp(UOW)
                                             uowstate(reset)
                                             uowid(unit of work ID)
         <------------------------------
SYNCRSY
  (Sync Point Resync)
  rsynctyp(end)
         ------------------------------>
                                           SYNCRRD
                                             (SYNCRSY Reply Data)
                                             rsynctyp(forget)
         <------------------------------
close(conversation)                        close(conversation)
```

Figure 3-102 Resync Connection (Part 2)

Cold Start Considerations

A cold start may occur at one or more of the SYNPTMGRs 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 3-103 on page 811 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.

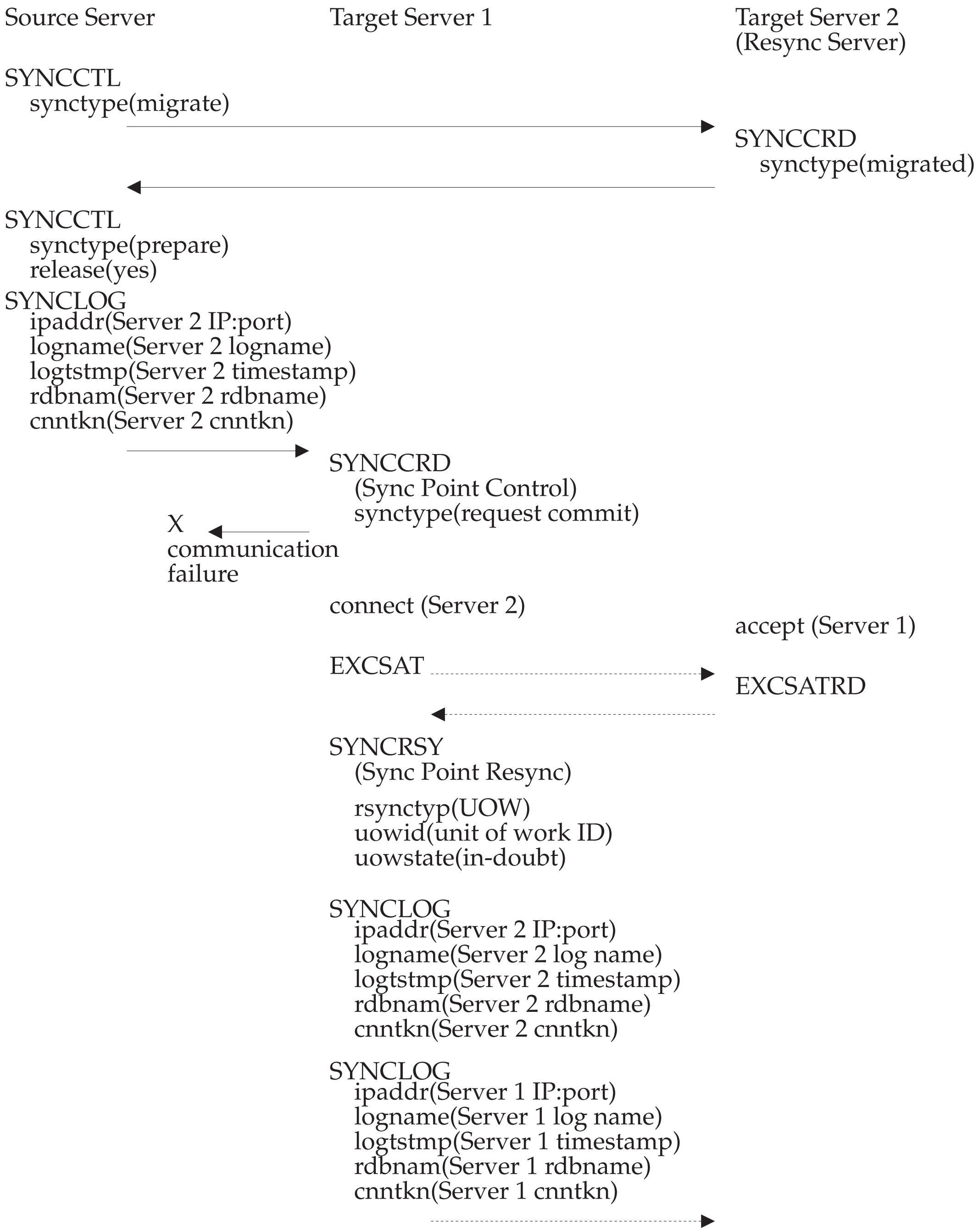


Figure 3-103 Multiple Connections Race Condition (Part 1)

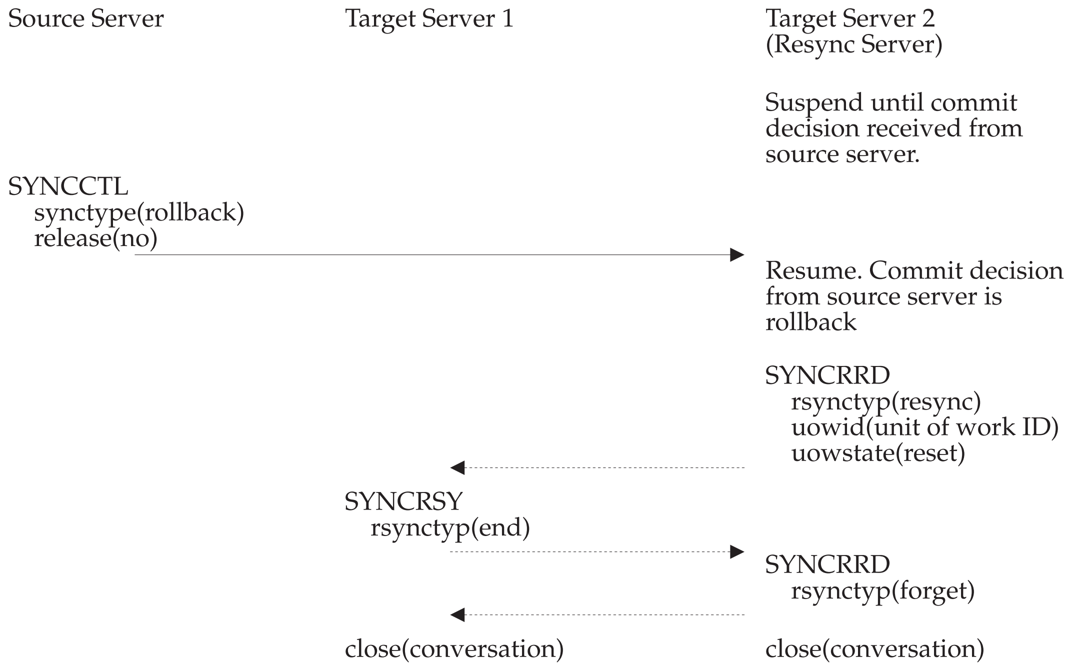


Figure 3-104 Multiple Connections Race Condition (Part 2)

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 that 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 SYNCPTRMGR (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 3-105 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 3-22 Coordinator SYNCPTMGR With Log

Events		States				
		reset	prepare	rollback	committed	resync
E1	Application issues commit.	(prepare: E4)				
E2	Application issue 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	

Events		States reset	prepare	rollback	committed	resync
					(committed: E8)	
E8	Receive forget or an implied forget from participants that voted request to commit to prepare.				ACF (resync: E11) ACF&REL -CLOSE -Write FLR (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	

Events		States				
		reset	prepared	forget	committed	resync
					-Write FLR (forget: E6) CF&RCT& EF -COMMIT -Write FLR (forget: E6) CF&REL& RCT&! EF -COMMIT -Write FLR (reset: E1) CF&RRB -ABORT -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

Events		States				
		reset	**prepared**	**forget**	**committed**	**resync**
						(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 3-23 Coordinator SYNCPTMGR With No Log

		States				
Events		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				CF -RSYNC (reset: E1)	

Events		States				
		reset	**migrate**	**rollback**	**prepare**	**committed**
	voted request to commit.				CF (prepare: E9)	
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 3-24 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)			

	Events	reset	prepare	rollback	committed	resync
		States				
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) 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 rollbacked.
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

Variable	Reference
insvar	*PRCCNVCD* on page 521
semantic	*LOGTSTMP* on page 411
	SYNCCTL on page 760
	SYNCPTMGR on page 782
	SYNCRSY on page 828

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'126D'	
type	INSTANCE_OF REQUIRED	RSYNCTYP - Resync Type
uowid	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier This parameter is required when the value of parameter *rsynctyp* is X'01'. Otherwise, it must not be present.
uowstate	INSTANCE_OF OPTIONAL NOTE	UOWSTATE - Unit of Work State This parameter is required when the value of parameter *rsynctyp* is X'01'. Otherwise, it must not be present.

SEE ALSO

Variable	Reference
rpydta	*SYNCRSY* on page 828
semantic	*SYNCPTOV* on page 787

SYNCRSY

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 SYNCPTOV for more information on target system processing and reply messages.

Exceptions

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1069'	
rsynctyp	INSTANCE_OF REQUIRED CMDTRG	RSYNCTYP - Resync Type

uowid	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier This parameter is required if *rsynctyp* 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 *rsynctyp* is set to UOW (X'01'). Otherwise this parameter must not be present.
clscmd	**NIL**	
inscmd	**NIL**	
cmddta		**COMMAND OBJECTS**
X'106F'	INSTANCE_OF OPTIONAL NOTE	SYNCLOG - Sync Point Log When the parameter *rsynctyp* 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.
rpydta		**REPLY OBJECTS**
X'126D'	INSTANCE_OF REQUIRED	SYNCRRD - Sync Point Resync Reply
cmdrpy		**COMMAND REPLIES**
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

Variable	Reference
insvar	*PRCCNVCD* on page 521
semantic	*SYNCPTOV* on page 787

SYNCTYPE

NAME

SYNCTYPE — Sync Point Operation Type

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1187'

Length *

Class CLASS

Sprcls SCALAR - Scalar Object

Syncpoint operation (SYNCTYPE) specifies the type of syncpoint operation to be performed by a SYNCPTMGR.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'1187'	
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'01' Prepare to commit
	ENUVAL	X'02' Migrate 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 unit of work
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*SYNCCTL* on page 760
insvar	*SYNCCRD* on page 759
	SYNCCTL on page 760
rpydta	*SYNCCTL* on page 760

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, a 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	X'09'
	NOTE	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.
value (continued)	ENUVAL	X'0A'
	NOTE	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	X'0B'
	NOTE	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	X'0C'
	NOTE	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	X'0D'
	NOTE	Object code point index not supported. For example, a code point of 8032 is encountered, but X'8' is a reserved code point index.
	ENUVAL	X'0E'
	NOTE	Required object not found. For example, a CLEAR command does not have a *filnam* parameter present, or a MODREC command is not followed by a RECORD command data object.
	ENUVAL	X'0F'
	NOTE	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	X'10'
	NOTE	Mutually exclusive objects present. For example, a CRTDIRF command specifies both a DCLNAM and FILNAM parameters.
value (continued)	ENUVAL	X'11'
	NOTE	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	X'12'
	NOTE	Duplicate object present. For example, a LSTFAT command has two FILNAM parameters specified.
	ENUVAL	X'13'
	NOTE	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	X'14'
	NOTE	Required value not found.
	ENUVAL	X'15'
	NOTE	Reserved value not allowed. For example, a INSRECEF command specified a RECCNT(0) parameter.
	ENUVAL	X'16'
	NOTE	DSS continuation less than or equal to two. For example, the length bytes for the DSS continuation have a value of one.
	ENUVAL	X'17'
	NOTE	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	X'18'
	NOTE	DSS chaining bit not b'1', but DSSFMT bit3 set to b'1'.
	ENUVAL	X'19'
	NOTE	Previous DSS indicated current DSS has the same request correlator, but the request correlators are not the same.
value (continued)	ENUVAL	X'1A'
	NOTE	DSS chaining bit not b'1', but error continuation requested.
	ENUVAL	X'1B'
	MINLVL	2
	NOTE	Mutually exclusive parameter values specified. For example, an OPEN command specified PRPSHD(TRUE) and FILSHR(READER).
	ENUVAL	X'1D'
	NOTE	Code point not valid command. For example, the first code point in RQSDSS either is not in the dictionary or is not a code point for a command.
	REQUIRED	
clscmd	**NIL**	

inscmd	**NIL**

SEE ALSO

Variable	Reference
insvar	*SYNTAXRM* on page 835
semantic	*SYNTAXRM* on page 835

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'124C'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
synerrcd	INSTANCE_OF REQUIRED	SYNERRCD - Syntax Error Code
reccnt	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
codpnt	INSTANCE_OF MINLVL OPTIONAL NOTE	CODPNT - Code Point 3 Specifies the code point of the object that caused the syntax error.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information

clscmd	**NIL**
inscmd	**NIL**

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
semantic	*AGNCMDPR* on page 60
	CMNTCPIP on page 209
	DSS on page 289
	TCPSRCER on page 859
	TCPTRGER on page 861

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

None.

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 the term *TCPIPOVR* on page 847 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 3-106.

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).

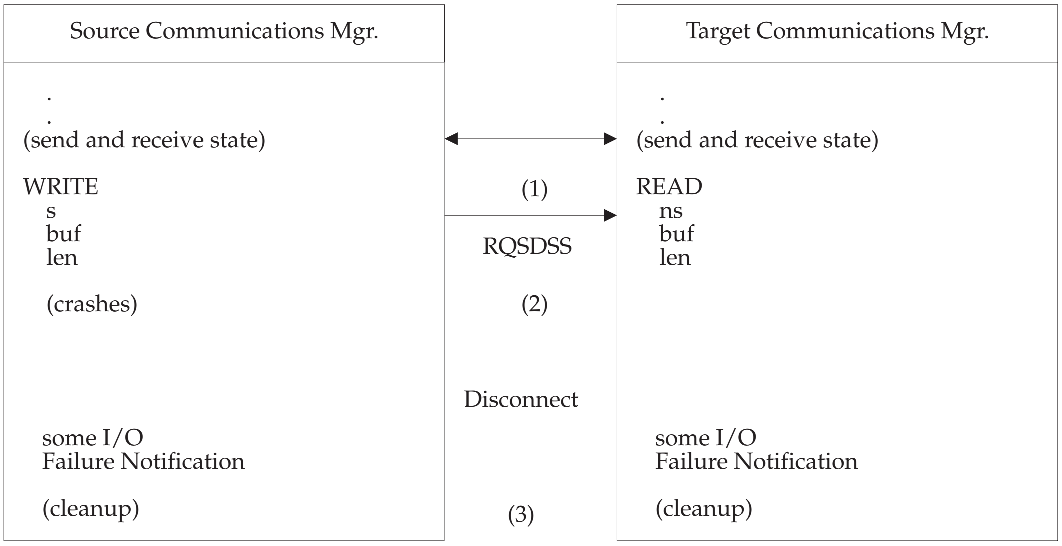


Figure 3-106 Communications Failure Between Source and Target

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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPCMNT on page 844
	TCPIPOVR on page 847

TCPCMNI

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 3-107 on page 841.

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).

Source Communications Mgr.		Target Communications Mgr.
SOCKET domain type protocol	(1)	SOCKET domain type protocol
s = socket (domain, type, protocol)	(2)	s = socket (domain, type, protocol)
	(3)	BIND s name namelen
	(4)	LISTEN s backlog
CONNECT s name namelen	(5) →	ACCEPT s addr addrlen
		ns = accept(s, addr, addrlen)
WRITE s buf len	(6) → EXCSAT (RQSDSS)	READ ns buf
READ s buf len	(7) ← EXCSATRD (OBJDSS)	WRITE ns buf len
WRITE s buf len	(8) → ACCSEC (RQSDSS)	READ ns buf
READ s buf len	(9) ← ACCSECRD (OBJDSS)	WRITE ns buf len
WRITE s buf len	(10) → SECCHK (RQSDSS)	READ ns buf
READ s buf len	(11) ← SECCHKRM (RPYDSS)	WRITE ns buf len
send and receive	↔ (12)	send and receive
WRITE s buf len	(13) → RQSDSS	READ ns buf len

Figure 3-107 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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPCMNFL on page 838
	TCPCMNT on page 844
	TCPIPOVR on page 847
	TCPSRCCD on page 853
	TCPSRCCR on page 856
	TCPSRCER on page 859
	TCPTRGER on page 861

TCPCMNT

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).
- No error situation occurs.
- Note that the full duplex environment allows SCM and TCM to send and receive messages simultaneously.

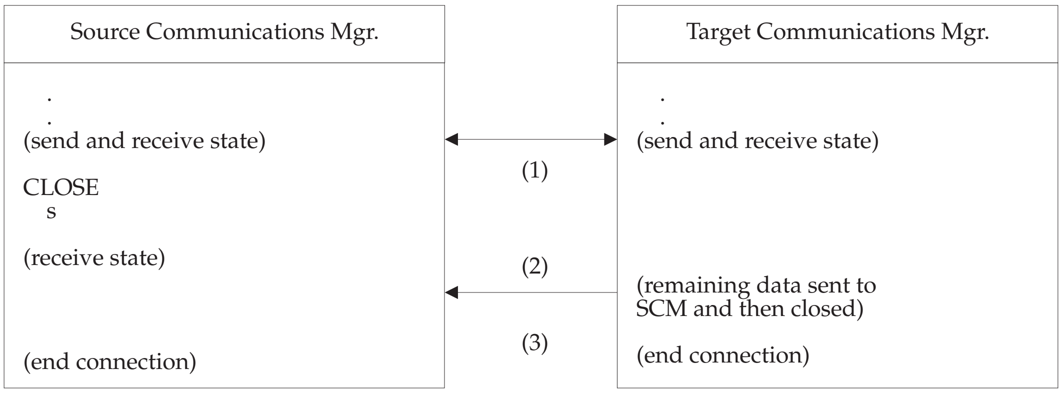


Figure 3-108 Normal Communications Termination

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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPIPOVR on page 847

TCPHOST

NAME

TCPHOST — TCP/IP Domain Qualified Host Name

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint 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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11DD'	
value	INSTANCE_OF MAXLEN	CHRSTRDR - Character String 256
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCLOG* on page 764

NAME

TCPIPOVR — TCP/IP Overview

DESCRIPTION (Semantic)

TCP/IP Overview

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 3-109.

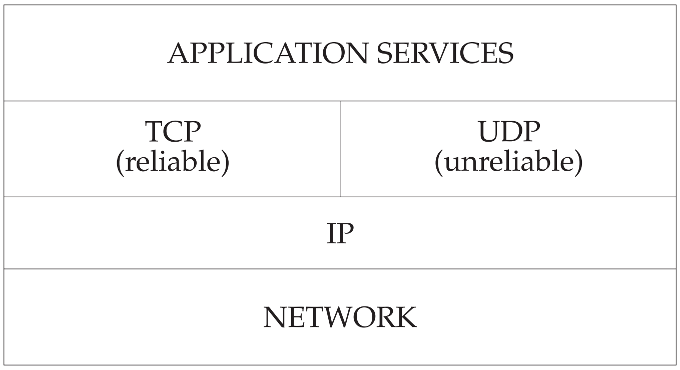


Figure 3-109 TCP/IP Internet Components

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 3-25 shows the bit structure of the four classes of Internet addresses.

Table 3-25 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 3-110 represents the structure of a class C address.

0 1 2	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3	3 4 5 6 7 8 9 0 1
1 1 0	Network	Local

Figure 3-110 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 (*CMNTCPIP* on page 209)

TCPCMNFL
TCP/IP Communications Failure (*TCPCMNFL* on page 838)

TCPCMNT
TCP/IP Communications Termination (*TCPCMNT* on page 844)

TCPSRCCD
TCP/IP Source Command with Data (*TCPSRCCD* on page 853)

TCPSRCCR
TCP/IP Source Command Returning Data (*TCPSRCCR* on page 856)

TCPSRCER
TCP/IP Source Detected Error (*TCPSRCER* on page 859)

TCPTRGER
TCP/IP Target Detected Error (*TCPTRGER* on page 861)

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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPCMNFL on page 838

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 3-111 on page 854. 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).
- The response to the RQSDSS is a single RPYDSS.
- No error situations occur.

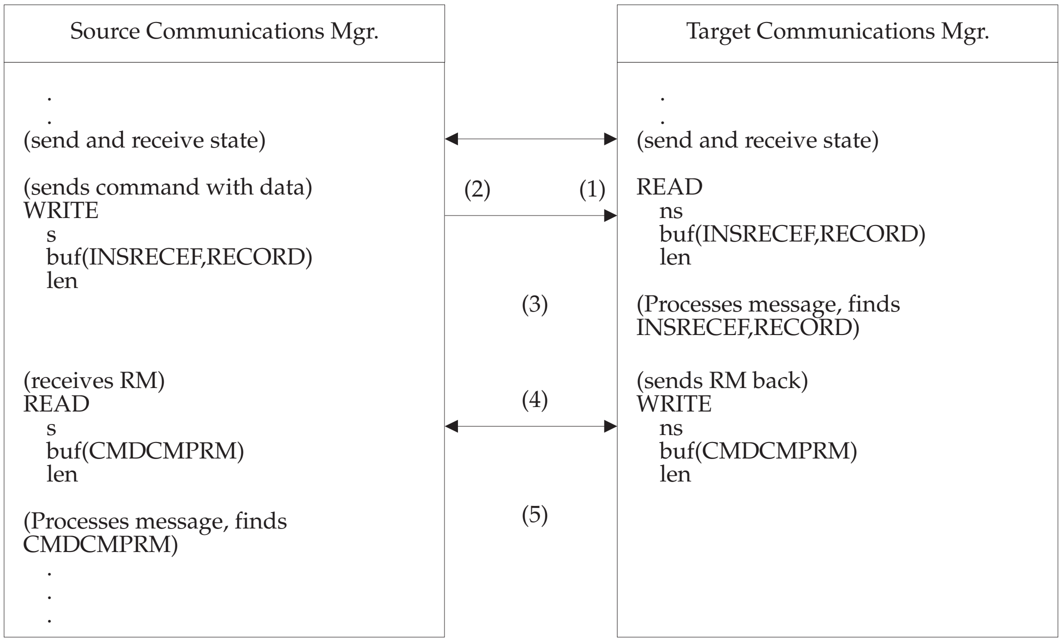


Figure 3-111 Source Sends Command With Data

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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPIPOVR on page 847

TCPSRCCR

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).
- No error situations occur.

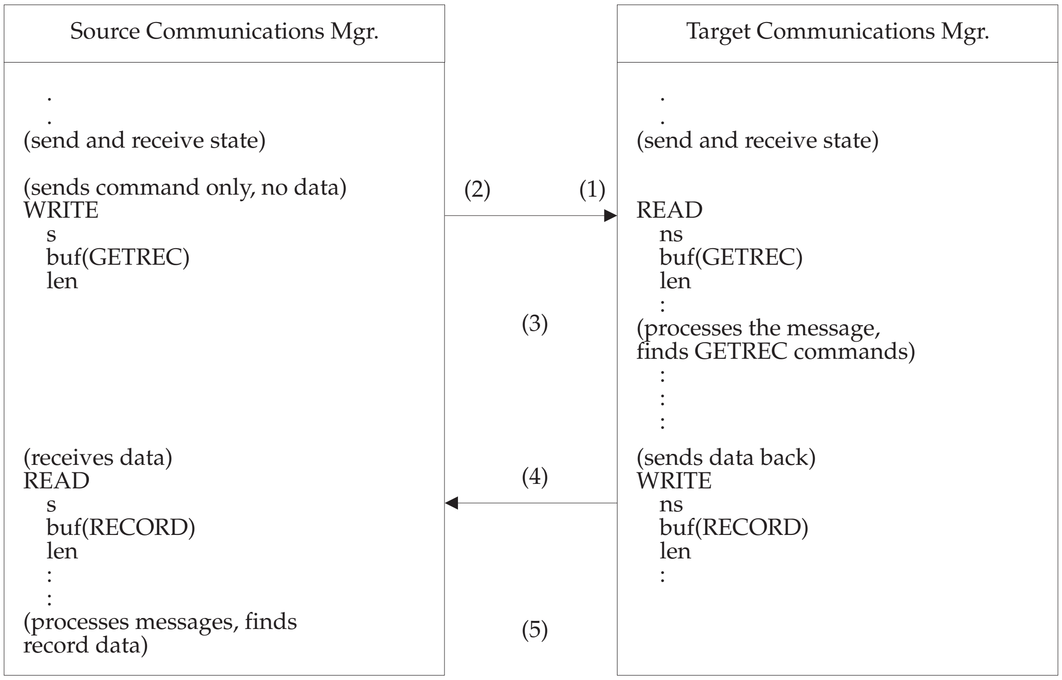


Figure 3-112 Source Sends Command With No Command Data

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.

TCPSRCCR

SEE ALSO

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPIPOVR on page 847

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 the description of the term TCPCMNI).
- 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.

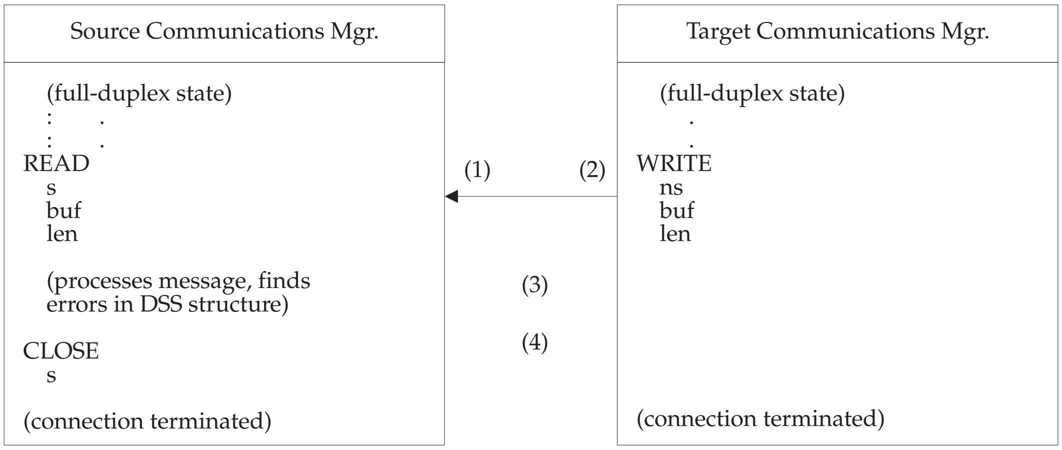


Figure 3-113 Source System Detects Error

Figure Notes

(1) The SCM's 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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPIPOVR on page 847

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).

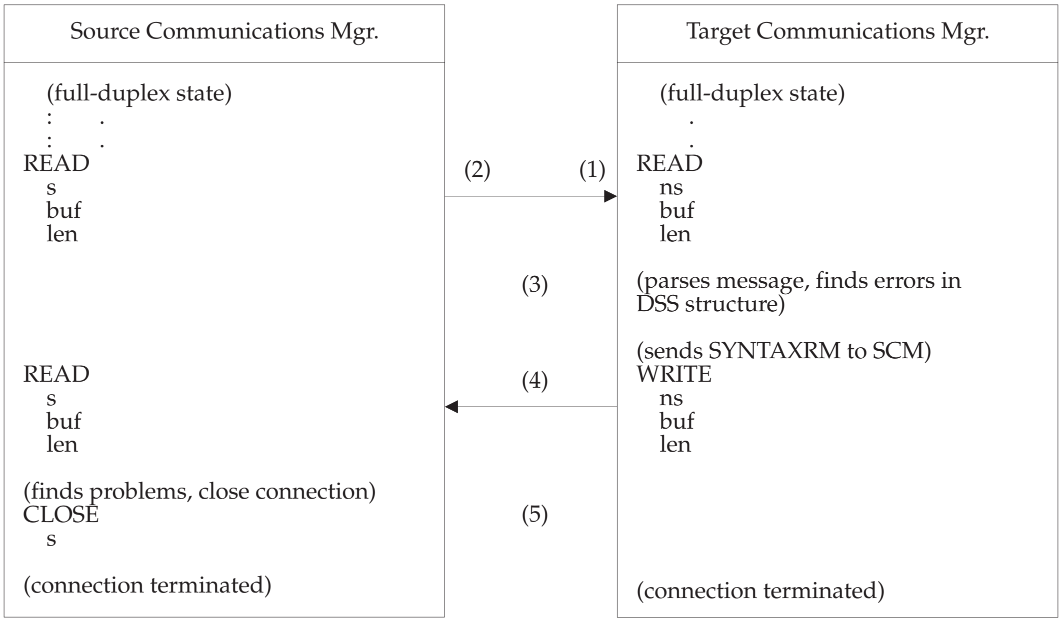


Figure 3-114 Target System Detects Error

Figure Notes

(1) The TCM's 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

Variable	Reference
semantic	*CMNTCPIP* on page 209
	TCPIPOVR on page 847
	TCPSRCER on page 859

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'114B'	
value	INSTANCE_OF MINLEN MAXLEN OPTIONAL	CHRSTRDR - Character String 0 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*CODPNT* on page 225
	CONSTANT on page 238
	HELP on page 371
sprcls	*TITLE* on page 865

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0045'	
value	INSTANCE_OF MINLEN MAXLEN OPTIONAL	CHRSTRDR - Character String 0 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
clsvar	*CLASS* on page 148
semantic	*BGNBND* on page 101
	CODPNT on page 225
	CONSTANT on page 238
	HELP on page 371
	BGNBND on page 101
	CLASS on page 148
	REBIND on page 606
	SCALAR on page 649
vldattls	*AGENT* on page 56

Variable	Reference
	CCSIDMGR on page 140
	CMNAPPC on page 179
	CMNMGR on page 191
	CMNSYNCPT on page 197
	CMNTCPIP on page 209
	DICTIONARY on page 272
	MANAGER on page 417
	RDB on page 571
	SECMGR on page 663
	SERVER on page 670
	SQLAM on page 694
	SUPERVISOR on page 753
	SYNCPTMGR on page 782

NAME

TRGDFTRT — Target Default Value Return

DESCRIPTION (Semantic)

Dictionary QDDRDBD

Codepoint X'213B'

Length *

Class CLASS

Sprcls BOOLEAN - Logical Value

Target Default Value Return (TRGDFTRT) Boolean State whether to return the target default values in ACCRDBRM when ACCRDB is successfully executed.

The value TRUE indicates that the target default values are returned.

The value FALSE indicates that the target default values are not returned.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'213B'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE DFTVAL	HEXSTRDR - Hexadecimal String 2 X'F1' - TRUE - True State Return the target default values. X'F0' - FALSE - False State Do not return the target default values. X'F0' - FALSE - False State
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDB* on page 42
	ACCRDBRM on page 48
semantic	*ACCRDB* on page 42
	ACCRDBRM on page 48

TRGNSPRM

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'125F'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
objnam	ENUCLS NOTE MINLVL ENUCLS NOTE MINLVL ENUCLS NOTE MINLVL ENUCLS NOTE MINLVL ENUCLS	RDBNAM - Relational Database Name The RDBNAM is returned when the target object that is not supported is an RDB. 3 FILNAM - File Name The FILNAM is returned when the target object that is not supported is a file. 4 DRCNAM - Directory Name The DRCNAM is returned when the target object that is not supported is a directory. 4 DCLNAM - Declared Name The DCLNAM is returned when the target object that is not supported is referenced by a declare name. 4 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	SRVDGN - Server Diagnostic Information
	OPTIONAL	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
semantic	*AGNCMDPR* on page 60
	SUBSETS on page 747

TRUE

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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	BOOLEAN on page 136
	OUTEXP on page 489
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDBALWUPD on page 580
	RDBCMTOK on page 585
	RLSCONV on page 619
	RTNSQLDA on page 648
	SQLCSRHLD on page 706
	SYNCCTL on page 760
	SYNERRCD on page 831
	TRGDFTRT on page 867
rpydta	*PRPSQLSTT* on page 533
	SYNCCTL on page 760
semantic	*ABBREVIATIONS* on page 30
	ACCRDBRM on page 48
	AGNCMDPR on page 60
	AGNRPYPR on page 63

Variable	Reference
	BOOLEAN on page 136
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OUTEXP on page 489
	QRYRELSCR on page 567
	QRYRFRTBL on page 568
	QRYROWNBR on page 569
	RDBALWUPD on page 580
	RTNSQLDA on page 648
	SQLCSRHLD on page 706
	SYNCCTL on page 760
	SYNCPTOV on page 787
	TRGDFTRT on page 867

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

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0029'	
typfmlnm	INSTANCE_OF OPTIONAL	TYPFMLNM - Data Type Family Name
fdodsc	INSTANCE_OF OPTIONAL	FDODSC - FD:OCA Data Descriptor
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*RDB* on page 571
	SQLAM on page 694
semantic	*SQLAM* on page 694
	TYPDEFNAM on page 873
	TYPDEFOVR on page 876
	TYPFMLNM on page 880
title	*TYPDEFOVR* on page 876

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 the description of the term TYPDEF.

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'002F'	
value	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
	REQUIRED	
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*BNDSQLSTT* on page 130
	DSCRDBTBL on page 281
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
insvar	*ACCRDB* on page 42
	ACCRDBRM on page 48
rpydta	*ACCRDB* on page 42
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
semantic	*ACCRDB* on page 42
	ACCRDBRM on page 48
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307

Variable	Reference
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	TYPDEF on page 872
	TYPDEFOVR on page 876

TYPDEFOVR

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 3-115 on page 877 illustrates TYPDEFOVR.

Timestamps	Commands/ Replies	CCSIDs specified	CCSIDs in effect
	ACCRDB/RM	SA DA MA	SA DA MA
	begin		
t1	COMMAND / REPLY	St1	St1 DA MA
t2		Dt2	St1 Dt2 MA
t3		St3 Mt3	St3 Dt2 Mt3
t4		Dt4	St3 Dt4 Mt3
	end		
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 3-115 The Usage of TYPDEFOVR

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0035'	
ccsidsbc	INSTANCE_OF OPTIONAL	CCSIDSBC - CCSID for Single-Byte Characters
ccsiddbc	INSTANCE_OF OPTIONAL	CCSIDDBC - CCSID for Double-byte Characters
ccsidmbc	INSTANCE_OF OPTIONAL	CCSIDMBC - CCSID for Mixed-byte Characters
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
cmddta	*BNDSQLSTT* on page 130
	DSCRDBTBL on page 281
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
insvar	*ACCRDB* on page 42
	ACCRDBRM on page 48
rpydta	*ACCRDB* on page 42
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
semantic	*ACCRDB* on page 42
	ACCRDBRM on page 48
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307

Variable	Reference
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	FIXROWPRC on page 365
	LMTBLKPRC on page 400
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606

TYPFMLNM

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 the description of the term TYPDEF.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'0030'	
value	INSTANCE_OF ENUVAL REQUIRED	NAMDR - Name Data 'QSQL'
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*TYPDEF* on page 872
semantic	*TYPDEF* on page 872

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.

UOWDSP

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	5	
class	X'2115'	
value	INSTANCE_OF LENGTH ENUVAL NOTE ENUVAL NOTE	HEXSTRDR - Hexadecimal String 2 01 Means the unit of work was committed. 02 Means the unit of work was rolled back.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ENDUOWRM* on page 314

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	29	
class	X'11AA'	
netname	INSTANCE_OF LENGTH REQUIRED	CHRSTRDR - Character String 17
instance	INSTANCE_OF LENGTH REQUIRED	BYTSTRDR - Byte String 6
sequence	INSTANCE_OF LENGTH REQUIRED	BINDR - Binary Number Field 16
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCCTL* on page 760
	SYNCRRD on page 826
	SYNCRSY on page 828
semantic	*SYNCCTL* on page 760
	SYNCPTOV on page 787

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.

clsvar	**NIL**	
insvar		**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
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*SYNCRRD* on page 826
	SYNCRSY on page 828
semantic	*SYNCPTOV* on page 787

USADATFMT

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

Variable	Reference
insvar	*STTDATFMT* on page 741
semantic	*ENDBND* on page 307

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 *Character Data Representation Architecture Registry* (SC09-1319, IBM).

SEE ALSO

Variable	Reference
insvar	*STTTIMFMT* on page 746
semantic	*ENDBND* on page 307

NAME

USRID — User ID at the Target System

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint 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.

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 the documentation in the *USRSECOVR* on page 893 term.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'11A0'	
value	INSTANCE_OF MINLEN MAXLEN REQUIRED	CHRSTRDR - Character String 1 255
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*ACCRDBRM* on page 48
	SECCHK on page 652
semantic	*ACCRDBRM* on page 48
	SECCHK on page 652
	USRSECOVR on page 893

NAME

USRIDNWPWD — Userid, Password, and New Password Security Mechanism

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Length *

Class CONSTANT

The Userid, 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:

USRIDSEC
User Identification Security Mechanism (*USRIDSEC* on page 892)

PWDSEC
Password Security Mechanism (*PWDSEC* on page 537)

NWPWDSEC
New Password Security Mechanism (*NWPWDSEC* on page 454)

USRSECOVR
User ID Security Overview (*USRSECOVR* on page 893)

value	5

SEE ALSO

Variable	Reference
insvar	*SECCHK* on page 652
	SECMEC on page 661
semantic	*SECCHK* on page 652
	SECMEC on page 661
	USRSECOVR on page 893

USRIDONL

NAME

USRIDONL — Userid Only Security Mechanism

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Length *

Class CONSTANT

The Userid 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 (*USRIDSEC* on page 892)

value	4

SEE ALSO

Variable	Reference
insvar	*SECCHK* on page 652
	SECMEC on page 661
semantic	*SECCHK* on page 652
	SECMEC on page 661

NAME

USRIDPWD — Userid and Password Security Mechanism

DESCRIPTION (Semantic)

Dictionary QDDPRMD

Length *

Class CONSTANT

The Userid 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:

USRIDSEC
User Identification Security Mechanism (*USRIDSEC* on page 892)

PWDSEC
Password Security Mechanism (*PWDSEC* on page 537)

USRSECOVR
User ID Security Overview (*USRSECOVR* on page 893)

value	3

SEE ALSO

Variable	Reference
insvar	*SECCHK* on page 652
	SECMEC on page 661
semantic	*SECCHK* on page 652
	SECMEC on page 661
	USRSECOVR on page 893

USRIDSEC

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

Variable	Reference
semantic	*SECMEC* on page 661
	USRIDNWPWD on page 889
	USRIDONL on page 890
	USRIDPWD on page 891
title	*USRIDNWPWD* on page 889
	USRIDONL on page 890
	USRIDPWD on page 891

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 (USRID) 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 3-116 indicates the DDM commands and replies that flow in the normal process of establishing a connection while using user ID and password security mechanisms.

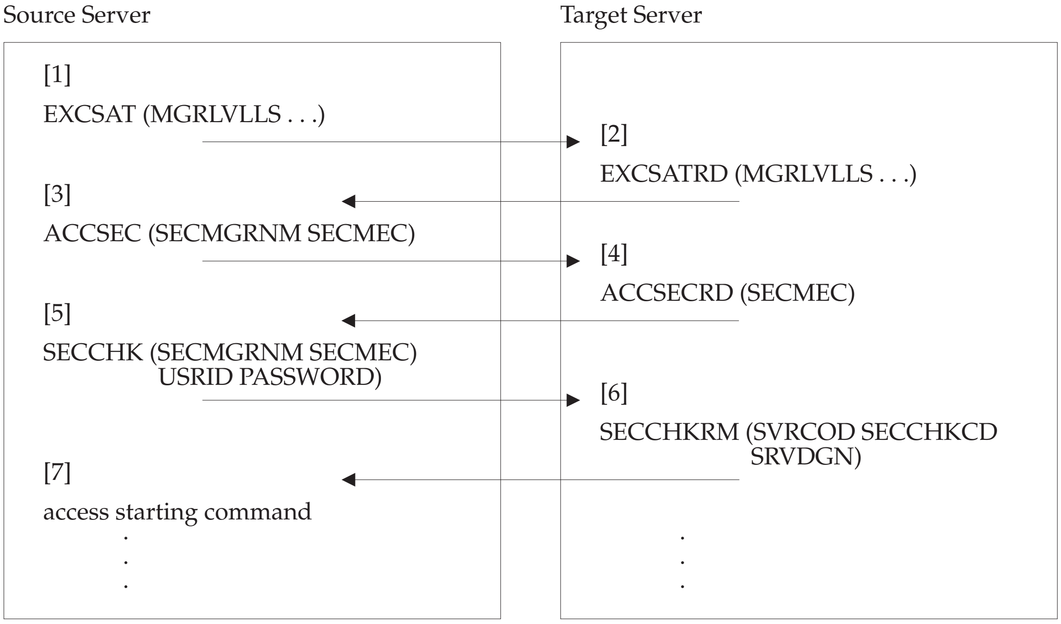


Figure 3-116 DDM USRID Security With a Password

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)
                           .
                           .
                           .))
```

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 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.

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 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.

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 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.

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 Security and New Password Security

Figure 3-117 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.

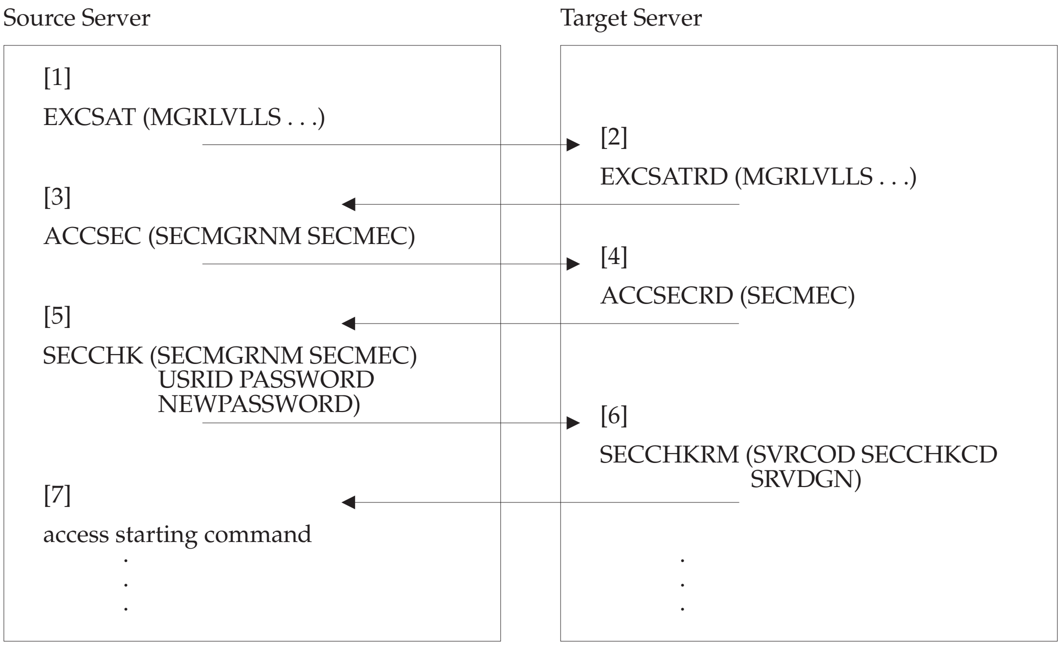


Figure 3-117 DDM USRID Security with a New Password

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.

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 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.

SEE ALSO

Variable	Reference
semantic	*NEWPASSWORD* on page 450
	NWPWDSEC on page 454
	PASSWORD on page 491
	PWDSEC on page 537
	SECMGR on page 663
	SECOVR on page 667
	USRID on page 888
	USRIDNWPWD on page 889
	USRIDPWD on page 891

VALNSPRM

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 the description of the term *SUBSETS* on page 747).

The code point of the command parameter in error is returned as a parameter in this message.

DSS Carrier: RPYDSS

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1252'	
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
codpnt	INSTANCE_OF REQUIRED NOTE	CODPNT - Code Point Return the code point of the parameter whose value is not supported.
reccnt	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
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
clscmd	**NIL**	

inscmd	**NIL**

SEE ALSO

Variable	Reference
cmdrpy	*ACCRDB* on page 42
	ACCSEC on page 51
	BGNBND on page 101
	BNDSQLSTT on page 130
	CLSQRY on page 163
	CNTQRY on page 217
	COMMAND on page 233
	DRPPKG on page 274
	DSCRDBTBL on page 281
	DSCSQLSTT on page 285
	ENDBND on page 307
	EXCSAT on page 323
	EXCSQLIMM on page 331
	EXCSQLSTT on page 336
	OPNQRY on page 475
	PRPSQLSTT on page 533
	RDBCMM on page 582
	RDBRLLBCK on page 598
	REBIND on page 606
	SECCHK on page 652
	SYNCCTL on page 760
	SYNCRSY on page 828
semantic	*ACCRDB* on page 42
	SUBSETS on page 747

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.

clsvar	**NIL**	
insvar		**CLASS INSTANCE VARIABLES**
length	*	
class	X'1144'	
value	INSTANCE_OF MAXLEN DFTVAL NOTE	NAMDR - Name Data 254 '' The default value means 'null'.
clscmd	**NIL**	
inscmd	**NIL**	

SEE ALSO

Variable	Reference
insvar	*BGNBND* on page 101
	BGNBNDRM on page 109
	DRPPKG on page 274
	PKGRPLVRS on page 518
	REBIND on page 606
semantic	*BGNBND* on page 101
	DRPPKG on page 274
	ENDBND on page 307
	PKGATHOPT on page 493
	PKGID on page 504
	PKGNAM on page 507
	PKGRPLALW on page 514
	PKGRPLNA on page 515

Variable	Reference
	PKGRPLOPT on page 516
	REBIND on page 606
	SQL on page 680

WARNING

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

Variable	Reference
insvar	*ACCRDBRM* on page 48
	CMDCHKRM on page 170
	CMDNSPRM on page 173
	ENDQRYRM on page 312
	ENDUOWRM on page 314
	QRYNOPRM on page 562
	RSCLMTRM on page 634
	SVRCOD on page 756
semantic	*ACCRDB* on page 42
	ACCRDBRM on page 48
	DCESECOVR on page 248

Appendix A

Code Points (Sorted by Term Name)

The following table lists the code points, sorted by term name:

Term Name	Code Point (Hex)	Reference
ABNUOWRM	220D	*ABNUOWRM* on page 39
ACCDMG	003E	*ACCDMG* on page 41
ACCRDB	2001	*ACCRDB* on page 42
ACCRDBRM	2201	*ACCRDBRM* on page 48
ACCSEC	106D	*ACCSEC* on page 51
ACCSECRD	14AC	*ACCSECRD* on page 55
AGENT	1403	*AGENT* on page 56
AGNPRMRM	1232	*AGNPRMRM* on page 61
ARRAY	004B	*ARRAY* on page 96
ASSOCIATION	0001	*ASSOCIATION* on page 98
ATTLST	0046	*ATTLST* on page 99
BGNBND	2002	*BGNBND* on page 101
BGNBNDRM	2208	*BGNBNDRM* on page 109
BIN	0003	*BIN* on page 110
BINDR	0042	*BINDR* on page 112
BITDR	0004	*BITDR* on page 115
BITSTRDR	0005	*BITSTRDR* on page 117
BNDCHKEXS	211B	*BNDCHKEXS* on page 119
BNDCHKONL	2421	*BNDCHKONL* on page 120
BNDCRTCTL	211D	*BNDCRTCTL* on page 121
BNDERRALW	2423	*BNDERRALW* on page 122
BNDEXPOPT	2130	*BNDEXPOPT* on page 123
BNDEXSOPT	241D	*BNDEXSOPT* on page 124
BNDEXSRQR	241C	*BNDEXSRQR* on page 125
BNDNERALW	2422	*BNDNERALW* on page 126
BNDOPT	2405	*BNDOPT* on page 127
BNDOPTNM	2144	*BNDOPTNM* on page 128
BNDOPTVL	2145	*BNDOPTVL* on page 129
BNDSQLSTT	2004	*BNDSQLSTT* on page 130
BNDSTTASM	2126	*BNDSTTASM* on page 135

Term Name	Code Point (Hex)	Reference
BOOLEAN	0006	*BOOLEAN* on page 136
BYTDR	0043	*BYTDR* on page 137
CCSIDDBC	119D	*CCSIDDBC* on page 138
CCSIDMBC	119E	*CCSIDMBC* on page 139
CCSIDMGR	14CC	*CCSIDMGR* on page 140
CCSIDSBC	119C	*CCSIDSBC* on page 144
CHRDR	0008	*CHRDR* on page 145
CHRSTRDR	0009	*CHRSTRDR* on page 146
CLASS	000A	*CLASS* on page 148
CLSQRY	2005	*CLSQRY* on page 163
CMDATHRM	121C	*CMDATHRM* on page 168
CMDCHKRM	1254	*CMDCHKRM* on page 170
CMDCMPRM	124B	*CMDCMPRM* on page 172
CMDNSPRM	1250	*CMDNSPRM* on page 173
CMDTRG	0041	*CMDTRG* on page 175
CMDVLTRM	221D	*CMDVLTRM* on page 176
CMMRQSRM	2225	*CMMRQSRM* on page 177
CMMTYP	2143	*CMMTYP* on page 178
CMNAPPC	1444	*CMNAPPC* on page 179
CMNMGR	1408	*CMNMGR* on page 191
CMNSYNCPT	147C	*CMNSYNCPT* on page 197
CMNTCPIP	1474	*CMNTCPIP* on page 209
CNNTKN	1070	*CNNTKN* on page 215
CNSVAL	000B	*CNSVAL* on page 216
CNTQRY	2006	*CNTQRY* on page 217
codpnt	000C	*CODPNT* on page 225
codpntDR	0064	*CODPNTDR* on page 228
COLLECTION	000D	*COLLECTION* on page 231
COMMAND	000E	*COMMAND* on page 233
CONSTANT	0050	*CONSTANT* on page 238
CRRTKN	2135	*CRRTKN* on page 240
CSTBITS	2433	*CSTBITS* on page 241
CSTMBCS	2435	*CSTMBCS* on page 242
CSTSBCS	2434	*CSTSBCS* on page 243
CSTSYSDFT	2432	*CSTSYSDFT* on page 244

Term Name	Code Point (Hex)	Reference
DATA	003C	*DATA* on page 245
DCTIND	1450	*DCTIND* on page 253
DCTINDEN	1451	*DCTINDEN* on page 254
DECDELCMA	243D	*DECDELCMA* on page 259
DECDELPRD	243C	*DECDELPRD* on page 260
DECPRC	2106	*DECPRC* on page 261
DEFINITION	0048	*DEFINITION* on page 262
DEFLST	0047	*DEFLST* on page 263
DEPERRCD	119B	*DEPERRCD* on page 265
DFTPKG	241E	*DFTPKG* on page 267
DFTRDBCOL	2128	*DFTRDBCOL* on page 268
DFTVAL	0011	*DFTVAL* on page 269
DGRIOPRL	212F	*DGRIOPRL* on page 270
DICTIONARY	1458	*DICTIONARY* on page 272
DRPPKG	2007	*DRPPKG* on page 274
DSCERRCD	2101	*DSCERRCD* on page 277
DSCINVRM	220A	*DSCINVRM* on page 279
DSCRDBTBL	2012	*DSCRDBTBL* on page 281
DSCSQLSTT	2008	*DSCSQLSTT* on page 285
DSSFMT	140D	*DSSFMT* on page 301
DTAMCHRM	220E	*DTAMCHRM* on page 303
ELMCLS	004D	*ELMCLS* on page 306
ENDBND	2009	*ENDBND* on page 307
ENDQRYRM	220B	*ENDQRYRM* on page 312
ENDUOWRM	220C	*ENDUOWRM* on page 314
ENUCLS	0040	*ENUCLS* on page 316
ENULEN	0015	*ENULEN* on page 317
ENUVAL	0016	*ENUVAL* on page 318
ERROR	0017	*ERROR* on page 319
EURDATFMT	242B	*EURDATFMT* on page 321
EURTIMFMT	2430	*EURTIMFMT* on page 322
EXCSAT	1041	*EXCSAT* on page 323
EXCSATRD	1443	*EXCSATRD* on page 329
EXCSQLIMM	200A	*EXCSQLIMM* on page 331
EXCSQLSTT	200B	*EXCSQLSTT* on page 336

Term Name	Code Point (Hex)	Reference
EXPALL	243B	*EXPALL* on page 344
EXPNON	243A	*EXPNON* on page 345
EXTNAM	115E	*EXTNAM* on page 348
FALSE	0018	*FALSE* on page 349
FDODSC	0010	*FDODSC* on page 354
FDODSCOFF	2118	*FDODSCOFF* on page 356
FDODTA	147A	*FDODTA* on page 357
FDODTAOFF	2119	*FDODTAOFF* on page 359
FDOOBJ	1480	*FDOOBJ* on page 360
FDOPRMOFF	212B	*FDOPRMOFF* on page 361
FDOTRPOFF	212A	*FDOTRPOFF* on page 362
FIELD	006A	*FIELD* on page 363
FIXROWPRC	2418	*FIXROWPRC* on page 365
FORGET	1186	*FORGET* on page 369
FRCFIXROW	2410	*FRCFIXROW* on page 370
HELP	0019	*HELP* on page 371
HEXDR	001A	*HEXDR* on page 374
HEXSTRDR	001B	*HEXSTRDR* on page 376
IGNORABLE	001C	*IGNORABLE* on page 378
INFO	001E	*INFO* on page 379
INHERITED	0049	*INHERITED* on page 385
INSTANCE_OF	005D	*INSTANCE_OF* on page 387
IPADDR	11E8	*IPADDR* on page 388
ISODATFMT	2429	*ISODATFMT* on page 389
ISOLVLALL	2443	*ISOLVLALL* on page 390
ISOLVLCHG	2441	*ISOLVLCHG* on page 391
ISOLVLCS	2442	*ISOLVLCS* on page 392
ISOLVLNC	2445	*ISOLVLNC* on page 393
ISOLVLRR	2444	*ISOLVLRR* on page 394
ISOTIMFMT	242E	*ISOTIMFMT* on page 395
JISDATFMT	242C	*JISDATFMT* on page 396
JISTIMFMT	2431	*JISTIMFMT* on page 397
LENGTH	001F	*LENGTH* on page 398
LMTBLKPRC	2417	*LMTBLKPRC* on page 400
LOGNAME	1184	*LOGNAME* on page 410

Term Name	Code Point (Hex)	Reference
LOGTSTMP	1185	*LOGTSTMP* on page 411
MANAGER	1456	*MANAGER* on page 417
MAXBLKEXT	2141	*MAXBLKEXT* on page 420
MAXLEN	0021	*MAXLEN* on page 422
MAXRSLCNT	2140	*MAXRSLCNT* on page 423
MAXSCTNBR	2127	*MAXSCTNBR* on page 424
MAXVAL	0022	*MAXVAL* on page 425
MGRDEPRM	1218	*MGRDEPRM* on page 426
MGRLVL	1442	*MGRLVL* on page 427
MGRLVLLS	1404	*MGRLVLLS* on page 429
mgrlvln	1473	*MGRLVLN* on page 430
MGRLVLRM	1210	*MGRLVLRM* on page 432
MGRNAM	1452	*MGRNAM* on page 434
MINLEN	0025	*MINLEN* on page 439
MINLVL	0002	*MINLVL* on page 440
MINVAL	0026	*MINVAL* on page 441
MTLEXC	0067	*MTLEXC* on page 442
MTLINC	00A7	*MTLINC* on page 443
NAMDR	0066	*NAMDR* on page 444
NAME	0027	*NAME* on page 445
NAMSYMDR	0061	*NAMSYMDR* on page 447
NBRROW	213A	*NBRROW* on page 449
NEWPASSWORD	11DE	*NEWPASSWORD* on page 450
NIL	002A	*NIL* on page 451
NOTE	0014	*NOTE* on page 452
NUMBER	002B	*NUMBER* on page 453
OBJDSS	1429	*OBJDSS* on page 455
OBJECT	002C	*OBJECT* on page 459
OBJNSPRM	1253	*OBJNSPRM* on page 462
OPNQFLRM	2212	*OPNQFLRM* on page 474
OPNQRY	200C	*OPNQRY* on page 475
OPNQRYRM	2205	*OPNQRYRM* on page 483
OPTIONAL	002D	*OPTIONAL* on page 485
ORDCOL	004C	*ORDCOL* on page 487
OUTEXP	2111	*OUTEXP* on page 489

Term Name	Code Point (Hex)	Reference
PASSWORD	11A1	*PASSWORD* on page 491
PKGATHKP	2425	*PKGATHKP* on page 492
PKGATHOPT	211E	*PKGATHOPT* on page 493
PKGATHRUL	213F	*PKGATHRUL* on page 494
PKGATHRVK	2424	*PKGATHRVK* on page 496
PKGBNARM	2206	*PKGBNARM* on page 497
PKGBPARM	2209	*PKGBPARM* on page 498
PKGCNSTKN	210D	*PKGCNSTKN* on page 500
PKGDFTCC	119A	*PKGDFTCC* on page 501
PKGDFTCST	2125	*PKGDFTCST* on page 503
PKGID	2109	*PKGID* on page 504
PKGISOLVL	2124	*PKGISOLVL* on page 505
PKGNAM	210A	*PKGNAM* on page 507
PKGNAMCSN	2113	*PKGNAMCSN* on page 508
PKGNAMCT	2112	*PKGNAMCT* on page 510
PKGOWNID	2131	*PKGOWNID* on page 512
PKGRPLALW	241F	*PKGRPLALW* on page 514
PKGRPLNA	2420	*PKGRPLNA* on page 515
PKGRPLOPT	211C	*PKGRPLOPT* on page 516
PKGRPLVRS	212D	*PKGRPLVRS* on page 518
PKGSN	210C	*PKGSN* on page 519
PKGSNLST	2139	*PKGSNLST* on page 520
PRCCNVCD	113F	*PRCCNVCD* on page 521
PRCCNVRM	1245	*PRCCNVRM* on page 523
PRCNAM	2138	*PRCNAM* on page 525
PRDDTA	2104	*PRDDTA* on page 528
PRDID	112E	*PRDID* on page 529
PRMDMG	002E	*PRMDMG* on page 530
PRMNSPRM	1251	*PRMNSPRM* on page 531
PRPSQLSTT	200D	*PRPSQLSTT* on page 533
QLFATT	0007	*QLFATT* on page 554
QRYBLKCTL	2132	*QRYBLKCTL* on page 557
QRYBLKSZ	2114	*QRYBLKSZ* on page 559
QRYDSC	241A	*QRYDSC* on page 560
QRYDTA	241B	*QRYDTA* on page 561

Term Name	Code Point (Hex)	Reference
QRYNOPRM	2202	*QRYNOPRM* on page 562
QRYPOPRM	220F	*QRYPOPRM* on page 564
QRYPRCTYP	2102	*QRYPRCTYP* on page 566
QRYRELSCR	213C	*QRYRELSCR* on page 567
QRYRFRTBL	213E	*QRYRFRTBL* on page 568
QRYROWNBR	213D	*QRYROWNBR* on page 569
RDB	240F	*RDB* on page 571
RDBACCCL	210F	*RDBACCCL* on page 577
RDBACCRM	2207	*RDBACCRM* on page 578
RDBAFLRM	221A	*RDBAFLRM* on page 579
RDBALWUPD	211A	*RDBALWUPD* on page 580
RDBATHRM	2203	*RDBATHRM* on page 581
RDBCMM	200E	*RDBCMM* on page 582
RDBCMTOK	2105	*RDBCMTOK* on page 585
RDBCOLID	2108	*RDBCOLID* on page 586
RDBNACRM	2204	*RDBNACRM* on page 587
RDBNAM	2110	*RDBNAM* on page 589
RDBNFNRM	2211	*RDBNFNRM* on page 592
RDBRLLBCK	200F	*RDBRLLBCK* on page 598
RDBRLSCMM	2438	*RDBRLSCMM* on page 601
RDBRLSCNV	2439	*RDBRLSCNV* on page 602
RDBRLSOPT	2129	*RDBRLSOPT* on page 603
RDBUPDRM	2218	*RDBUPDRM* on page 604
REBIND	2010	*REBIND* on page 606
REPEATABLE	0031	*REPEATABLE* on page 612
REQUIRED	0032	*REQUIRED* on page 615
RESERVED	0033	*RESERVED* on page 617
RLSCONV	119F	*RLSCONV* on page 619
RPYDSS	1436	*RPYDSS* on page 620
RPYMSG	1437	*RPYMSG* on page 624
RQSCRR	1438	*RQSCRR* on page 626
RQSDSS	1439	*RQSDSS* on page 629
RSCLMTRM	1233	*RSCLMTRM* on page 634
RSCNAM	112D	*RSCNAM* on page 637
RSCTYP	111F	*RSCTYP* on page 638

Term Name	Code Point (Hex)	Reference
RSLSETFLG	2142	*RSLSETFLG* on page 639
RSLSETRM	2219	*RSLSETRM* on page 642
RSNCOD	1127	*RSNCOD* on page 643
RSYNCMGR	14C1	*RSYNCMGR* on page 644
RSYNCTYP	11EA	*RSYNCTYP* on page 647
RTNSQLDA	2116	*RTNSQLDA* on page 648
SCALAR	0034	*SCALAR* on page 649
SECCHK	106E	*SECCHK* on page 652
SECCHKCD	11A4	*SECCHKCD* on page 656
SECCHKRM	1219	*SECCHKRM* on page 659
SECMEC	11A2	*SECMEC* on page 661
SECMGR	1440	*SECMGR* on page 663
SECMGRNM	1196	*SECMGRNM* on page 666
SECTKN	11DC	*SECTKN* on page 669
SERVER	1448	*SERVER* on page 670
SESDMG	003F	*SESDMG* on page 672
SEVERE	003A	*SEVERE* on page 673
SNAADDR	11E9	*SNAADDR* on page 674
SPCVAL	0036	*SPCVAL* on page 676
SPRCLS	0037	*SPRCLS* on page 678
SPVNAM	115D	*SPVNAM* on page 679
SQLAM	2407	*SQLAM* on page 694
SQLCARD	2408	*SQLCARD* on page 702
SQLCINRD	240B	*SQLCINRD* on page 705
SQLCSRHLD	211F	*SQLCSRHLD* on page 706
SQLDARD	2411	*SQLDARD* on page 707
SQLDTA	2412	*SQLDTA* on page 708
SQLDTARD	2413	*SQLDTARD* on page 709
SQLERRRM	2213	*SQLERRRM* on page 710
SQLOBJNAM	243E	*SQLOBJNAM* on page 712
SQLRSLRD	240E	*SQLRSLRD* on page 713
SQLSTT	2414	*SQLSTT* on page 714
SQLSTTNBR	2117	*SQLSTTNBR* on page 715
SQLSTTVRB	2419	*SQLSTTVRB* on page 716
SRVCLSNM	1147	*SRVCLSNM* on page 717

Term Name	Code Point (Hex)	Reference
SRVDGN	1153	*SRVDGN* on page 720
SRVLCNT	244C	*SRVLCNT* on page 722
SRVLSRV	244D	*SRVLSRV* on page 723
SRVLST	244E	*SRVLST* on page 725
SRVNAM	116D	*SRVNAM* on page 727
SRVPRTY	244F	*SRVPRTY* on page 730
SRVRLSLV	115A	*SRVRLSLV* on page 731
STGLMT	1409	*STGLMT* on page 732
STRDELAP	2426	*STRDELAP* on page 733
STRDELDQ	2427	*STRDELDQ* on page 734
STRING	0038	*STRING* on page 735
STTASMEUI	2437	*STTASMEUI* on page 740
STTDATFMT	2122	*STTDATFMT* on page 741
STTDECDEL	2121	*STTDECDEL* on page 742
STTSCCCLS	2436	*STTSCCCLS* on page 743
STTSTRDEL	2120	*STTSTRDEL* on page 744
STTTIMFMT	2123	*STTTIMFMT* on page 746
SUPERVISOR	143C	*SUPERVISOR* on page 753
SVCERRNO	11B4	*SVCERRNO* on page 755
SVRCOD	1149	*SVRCOD* on page 756
SYNCCRD	1248	*SYNCCRD* on page 759
SYNCCTL	1055	*SYNCCTL* on page 760
SYNCLOG	106F	*SYNCLOG* on page 764
SYNCPTMGR	14C0	*SYNCPTMGR* on page 782
SYNCRRD	126D	*SYNCRRD* on page 826
SYNCRSY	1069	*SYNCRSY* on page 828
SYNCTYPE	1187	*SYNCTYPE* on page 830
SYNERRCD	114A	*SYNERRCD* on page 831
SYNTAXRM	124C	*SYNTAXRM* on page 835
TCPHOST	11DD	*TCPHOST* on page 846
TEXT	114B	*TEXT* on page 864
TITLE	0045	*TITLE* on page 865
TRGDFTRT	213B	*TRGDFTRT* on page 867
TRGNSPRM	125F	*TRGNSPRM* on page 868
TRUE	003B	*TRUE* on page 870

Term Name	Code Point (Hex)	Reference
TYPDEF	0029	*TYPDEF* on page 872
TYPDEFNAM	002F	*TYPDEFNAM* on page 873
TYPDEFOVR	0035	*TYPDEFOVR* on page 876
TYPFMLNM	0030	*TYPFMLNM* on page 880
UOWDSP	2115	*UOWDSP* on page 882
UOWID	11AA	*UOWID* on page 883
UOWSTATE	11AC	*UOWSTATE* on page 885
USADATFMT	242A	*USADATFMT* on page 886
USATIMFMT	242F	*USATIMFMT* on page 887
USRID	11A0	*USRID* on page 888
VALNSPRM	1252	*VALNSPRM* on page 898
VRSNAM	1144	*VRSNAM* on page 900
WARNING	003D	*WARNING* on page 902

Appendix B

Term Names (Sorted by Code Points)

The following table lists the term name, sorted by code points:

Code Point (Hex)	**Term Name**	**Reference**
0001	ASSOCIATION	*ASSOCIATION* on page 98
0002	MINLVL	*MINLVL* on page 440
0003	BIN	*BIN* on page 110
0004	BITDR	*BITDR* on page 115
0005	BITSTRDR	*BITSTRDR* on page 117
0006	BOOLEAN	*BOOLEAN* on page 136
0007	QLFATT	*QLFATT* on page 554
0008	CHRDR	*CHRDR* on page 145
0009	CHRSTRDR	*CHRSTRDR* on page 146
000A	CLASS	*CLASS* on page 148
000B	CNSVAL	*CNSVAL* on page 216
000C	codpnt	*CODPNT* on page 225
000D	COLLECTION	*COLLECTION* on page 231
000E	COMMAND	*COMMAND* on page 233
0010	FDODSC	*FDODSC* on page 354
0011	DFTVAL	*DFTVAL* on page 269
0014	NOTE	*NOTE* on page 452
0015	ENULEN	*ENULEN* on page 317
0016	ENUVAL	*ENUVAL* on page 318
0017	ERROR	*ERROR* on page 319
0018	FALSE	*FALSE* on page 349
0019	HELP	*HELP* on page 371
001A	HEXDR	*HEXDR* on page 374
001B	HEXSTRDR	*HEXSTRDR* on page 376
001C	IGNORABLE	*IGNORABLE* on page 378
001E	INFO	*INFO* on page 379
001F	LENGTH	*LENGTH* on page 398
0021	MAXLEN	*MAXLEN* on page 422
0022	MAXVAL	*MAXVAL* on page 425
0025	MINLEN	*MINLEN* on page 439

Code Point (Hex)	Term Name	Reference
0026	MINVAL	*MINVAL* on page 441
0027	NAME	*NAME* on page 445
0029	TYPDEF	*TYPDEF* on page 872
002A	**NIL**	*NIL* on page 451
002B	NUMBER	*NUMBER* on page 453
002C	OBJECT	*OBJECT* on page 459
002D	OPTIONAL	*OPTIONAL* on page 485
002E	PRMDMG	*PRMDMG* on page 530
002F	TYPDEFNAM	*TYPDEFNAM* on page 873
0030	TYPFMLNM	*TYPFMLNM* on page 880
0031	REPEATABLE	*REPEATABLE* on page 612
0032	REQUIRED	*REQUIRED* on page 615
0033	RESERVED	*RESERVED* on page 617
0034	SCALAR	*SCALAR* on page 649
0035	TYPDEFOVR	*TYPDEFOVR* on page 876
0036	SPCVAL	*SPCVAL* on page 676
0037	SPRCLS	*SPRCLS* on page 678
0038	STRING	*STRING* on page 735
003A	SEVERE	*SEVERE* on page 673
003B	TRUE	*TRUE* on page 870
003C	DATA	*DATA* on page 245
003D	WARNING	*WARNING* on page 902
003E	ACCDMG	*ACCDMG* on page 41
003F	SESDMG	*SESDMG* on page 672
0040	ENUCLS	*ENUCLS* on page 316
0041	CMDTRG	*CMDTRG* on page 175
0042	BINDR	*BINDR* on page 112
0043	BYTDR	*BYTDR* on page 137
0045	TITLE	*TITLE* on page 865
0046	ATTLST	*ATTLST* on page 99
0047	DEFLST	*DEFLST* on page 263
0048	DEFINITION	*DEFINITION* on page 262
0049	INHERITED	*INHERITED* on page 385
004B	ARRAY	*ARRAY* on page 96
004C	ORDCOL	*ORDCOL* on page 487

Code Point (Hex)	Term Name	Reference
004D	ELMCLS	*ELMCLS* on page 306
0050	CONSTANT	*CONSTANT* on page 238
005D	INSTANCE_OF	*INSTANCE_OF* on page 387
0061	NAMSYMDR	*NAMSYMDR* on page 447
0064	codpntDR	*CODPNTDR* on page 228
0066	NAMDR	*NAMDR* on page 444
0067	MTLEXC	*MTLEXC* on page 442
006A	FIELD	*FIELD* on page 363
00A7	MTLINC	*MTLINC* on page 443
1041	EXCSAT	*EXCSAT* on page 323
1055	SYNCCTL	*SYNCCTL* on page 760
1069	SYNCRSY	*SYNCRSY* on page 828
106D	ACCSEC	*ACCSEC* on page 51
106E	SECCHK	*SECCHK* on page 652
106F	SYNCLOG	*SYNCLOG* on page 764
1070	CNNTKN	*CNNTKN* on page 215
111F	RSCTYP	*RSCTYP* on page 638
1127	RSNCOD	*RSNCOD* on page 643
112D	RSCNAM	*RSCNAM* on page 637
112E	PRDID	*PRDID* on page 529
113F	PRCCNVCD	*PRCCNVCD* on page 521
1144	VRSNAM	*VRSNAM* on page 900
1147	SRVCLSNM	*SRVCLSNM* on page 717
1149	SVRCOD	*SVRCOD* on page 756
114A	SYNERRCD	*SYNERRCD* on page 831
114B	TEXT	*TEXT* on page 864
1153	SRVDGN	*SRVDGN* on page 720
115A	SRVRLSLV	*SRVRLSLV* on page 731
115D	SPVNAM	*SPVNAM* on page 679
115E	EXTNAM	*EXTNAM* on page 348
116D	SRVNAM	*SRVNAM* on page 727
1184	LOGNAME	*LOGNAME* on page 410
1185	LOGTSTMP	*LOGTSTMP* on page 411
1186	FORGET	*FORGET* on page 369
1187	SYNCTYPE	*SYNCTYPE* on page 830

Code Point (Hex)	Term Name	Reference
1196	SECMGRNM	*SECMGRNM* on page 666
119A	PKGDFTCC	*PKGDFTCC* on page 501
119B	DEPERRCD	*DEPERRCD* on page 265
119C	CCSIDSBC	*CCSIDSBC* on page 144
119D	CCSIDDBC	*CCSIDDBC* on page 138
119E	CCSIDMBC	*CCSIDMBC* on page 139
119F	RLSCONV	*RLSCONV* on page 619
11A0	USRID	*USRID* on page 888
11A1	PASSWORD	*PASSWORD* on page 491
11A2	SECMEC	*SECMEC* on page 661
11A4	SECCHKCD	*SECCHKCD* on page 656
11AA	UOWID	*UOWID* on page 883
11AC	UOWSTATE	*UOWSTATE* on page 885
11B4	SVCERRNO	*SVCERRNO* on page 755
11DC	SECTKN	*SECTKN* on page 669
11DD	TCPHOST	*TCPHOST* on page 846
11DE	NEWPASSWORD	*NEWPASSWORD* on page 450
11E8	IPADDR	*IPADDR* on page 388
11E9	SNAADDR	*SNAADDR* on page 674
11EA	RSYNCTYP	*RSYNCTYP* on page 647
1210	MGRLVLRM	*MGRLVLRM* on page 432
1218	MGRDEPRM	*MGRDEPRM* on page 426
1219	SECCHKRM	*SECCHKRM* on page 659
121C	CMDATHRM	*CMDATHRM* on page 168
1232	AGNPRMRM	*AGNPRMRM* on page 61
1233	RSCLMTRM	*RSCLMTRM* on page 634
1245	PRCCNVRM	*PRCCNVRM* on page 523
1248	SYNCCRD	*SYNCCRD* on page 759
124B	CMDCMPRM	*CMDCMPRM* on page 172
124C	SYNTAXRM	*SYNTAXRM* on page 835
1250	CMDNSPRM	*CMDNSPRM* on page 173
1251	PRMNSPRM	*PRMNSPRM* on page 531
1252	VALNSPRM	*VALNSPRM* on page 898
1253	OBJNSPRM	*OBJNSPRM* on page 462
1254	CMDCHKRM	*CMDCHKRM* on page 170

Code Point (Hex)	Term Name	Reference
125F	TRGNSPRM	*TRGNSPRM* on page 868
126D	SYNCRRD	*SYNCRRD* on page 826
1403	AGENT	*AGENT* on page 56
1404	MGRLVLLS	*MGRLVLLS* on page 429
1408	CMNMGR	*CMNMGR* on page 191
1409	STGLMT	*STGLMT* on page 732
140D	DSSFMT	*DSSFMT* on page 301
1429	OBJDSS	*OBJDSS* on page 455
1436	RPYDSS	*RPYDSS* on page 620
1437	RPYMSG	*RPYMSG* on page 624
1438	RQSCRR	*RQSCRR* on page 626
1439	RQSDSS	*RQSDSS* on page 629
143C	SUPERVISOR	*SUPERVISOR* on page 753
1440	SECMGR	*SECMGR* on page 663
1442	MGRLVL	*MGRLVL* on page 427
1443	EXCSATRD	*EXCSATRD* on page 329
1444	CMNAPPC	*CMNAPPC* on page 179
1448	SERVER	*SERVER* on page 670
1450	DCTIND	*DCTIND* on page 253
1451	DCTINDEN	*DCTINDEN* on page 254
1452	MGRNAM	*MGRNAM* on page 434
1456	MANAGER	*MANAGER* on page 417
1458	DICTIONARY	*DICTIONARY* on page 272
1473	**mgrlvln**	*MGRLVLN* on page 430
1474	CMNTCPIP	*CMNTCPIP* on page 209
147A	FDODTA	*FDODTA* on page 357
147C	CMNSYNCPT	*CMNSYNCPT* on page 197
1480	FDOOBJ	*FDOOBJ* on page 360
14AC	ACCSECRD	*ACCSECRD* on page 55
14C0	SYNCPTMGR	*SYNCPTMGR* on page 782
14C1	RSYNCMGR	*RSYNCMGR* on page 644
14CC	CCSIDMGR	*CCSIDMGR* on page 140
2001	ACCRDB	*ACCRDB* on page 42
2002	BGNBND	*BGNBND* on page 101
2004	BNDSQLSTT	*BNDSQLSTT* on page 130

Code Point (Hex)	Term Name	Reference
2005	CLSQRY	*CLSQRY* on page 163
2006	CNTQRY	*CNTQRY* on page 217
2007	DRPPKG	*DRPPKG* on page 274
2008	DSCSQLSTT	*DSCSQLSTT* on page 285
2009	ENDBND	*ENDBND* on page 307
200A	EXCSQLIMM	*EXCSQLIMM* on page 331
200B	EXCSQLSTT	*EXCSQLSTT* on page 336
200C	OPNQRY	*OPNQRY* on page 475
200D	PRPSQLSTT	*PRPSQLSTT* on page 533
200E	RDBCMM	*RDBCMM* on page 582
200F	RDBRLLBCK	*RDBRLLBCK* on page 598
2010	REBIND	*REBIND* on page 606
2012	DSCRDBTBL	*DSCRDBTBL* on page 281
2101	DSCERRCD	*DSCERRCD* on page 277
2102	QRYPRCTYP	*QRYPRCTYP* on page 566
2104	PRDDTA	*PRDDTA* on page 528
2105	RDBCMTOK	*RDBCMTOK* on page 585
2106	DECPRC	*DECPRC* on page 261
2108	RDBCOLID	*RDBCOLID* on page 586
2109	PKGID	*PKGID* on page 504
210A	PKGNAM	*PKGNAM* on page 507
210C	PKGSN	*PKGSN* on page 519
210D	PKGCNSTKN	*PKGCNSTKN* on page 500
210F	RDBACCCL	*RDBACCCL* on page 577
2110	RDBNAM	*RDBNAM* on page 589
2111	OUTEXP	*OUTEXP* on page 489
2112	PKGNAMCT	*PKGNAMCT* on page 510
2113	PKGNAMCSN	*PKGNAMCSN* on page 508
2114	QRYBLKSZ	*QRYBLKSZ* on page 559
2115	UOWDSP	*UOWDSP* on page 882
2116	RTNSQLDA	*RTNSQLDA* on page 648
2117	SQLSTTNBR	*SQLSTTNBR* on page 715
2118	FDODSCOFF	*FDODSCOFF* on page 356
2119	FDODTAOFF	*FDODTAOFF* on page 359
211A	RDBALWUPD	*RDBALWUPD* on page 580

Code Point (Hex)	Term Name	Reference
211B	BNDCHKEXS	*BNDCHKEXS* on page 119
211C	PKGRPLOPT	*PKGRPLOPT* on page 516
211D	BNDCRTCTL	*BNDCRTCTL* on page 121
211E	PKGATHOPT	*PKGATHOPT* on page 493
211F	SQLCSRHLD	*SQLCSRHLD* on page 706
2120	STTSTRDEL	*STTSTRDEL* on page 744
2121	STTDECDEL	*STTDECDEL* on page 742
2122	STTDATFMT	*STTDATFMT* on page 741
2123	STTTIMFMT	*STTTIMFMT* on page 746
2124	PKGISOLVL	*PKGISOLVL* on page 505
2125	PKGDFTCST	*PKGDFTCST* on page 503
2126	BNDSTTASM	*BNDSTTASM* on page 135
2127	MAXSCTNBR	*MAXSCTNBR* on page 424
2128	DFTRDBCOL	*DFTRDBCOL* on page 268
2129	RDBRLSOPT	*RDBRLSOPT* on page 603
212A	FDOTRPOFF	*FDOTRPOFF* on page 362
212B	FDOPRMOFF	*FDOPRMOFF* on page 361
212D	PKGRPLVRS	*PKGRPLVRS* on page 518
212F	DGRIOPRL	*DGRIOPRL* on page 270
2130	BNDEXPOPT	*BNDEXPOPT* on page 123
2131	PKGOWNID	*PKGOWNID* on page 512
2132	QRYBLKCTL	*QRYBLKCTL* on page 557
2135	CRRTKN	*CRRTKN* on page 240
2138	PRCNAM	*PRCNAM* on page 525
2139	PKGSNLST	*PKGSNLST* on page 520
213A	NBRROW	*NBRROW* on page 449
213B	TRGDFTRT	*TRGDFTRT* on page 867
213C	QRYRELSCR	*QRYRELSCR* on page 567
213D	QRYROWNBR	*QRYROWNBR* on page 569
213E	QRYRFRTBL	*QRYRFRTBL* on page 568
213F	PKGATHRUL	*PKGATHRUL* on page 494
2140	MAXRSLCNT	*MAXRSLCNT* on page 423
2141	MAXBLKEXT	*MAXBLKEXT* on page 420
2142	RSLSETFLG	*RSLSETFLG* on page 639
2143	CMMTYP	*CMMTYP* on page 178

Code Point (Hex)	Term Name	Reference
2144	BNDOPTNM	*BNDOPTNM* on page 128
2145	BNDOPTVL	*BNDOPTVL* on page 129
2201	ACCRDBRM	*ACCRDBRM* on page 48
2202	QRYNOPRM	*QRYNOPRM* on page 562
2203	RDBATHRM	*RDBATHRM* on page 581
2204	RDBNACRM	*RDBNACRM* on page 587
2205	OPNQRYRM	*OPNQRYRM* on page 483
2206	PKGBNARM	*PKGBNARM* on page 497
2207	RDBACCRM	*RDBACCRM* on page 578
2208	BGNBNDRM	*BGNBNDRM* on page 109
2209	PKGBPARM	*PKGBPARM* on page 498
220A	DSCINVRM	*DSCINVRM* on page 279
220B	ENDQRYRM	*ENDQRYRM* on page 312
220C	ENDUOWRM	*ENDUOWRM* on page 314
220D	ABNUOWRM	*ABNUOWRM* on page 39
220E	DTAMCHRM	*DTAMCHRM* on page 303
220F	QRYPOPRM	*QRYPOPRM* on page 564
2211	RDBNFNRM	*RDBNFNRM* on page 592
2212	OPNQFLRM	*OPNQFLRM* on page 474
2213	SQLERRRM	*SQLERRRM* on page 710
2218	RDBUPDRM	*RDBUPDRM* on page 604
2219	RSLSETRM	*RSLSETRM* on page 642
221A	RDBAFLRM	*RDBAFLRM* on page 579
221D	CMDVLTRM	*CMDVLTRM* on page 176
2225	CMMRQSRM	*CMMRQSRM* on page 177
2405	BNDOPT	*BNDOPT* on page 127
2407	SQLAM	*SQLAM* on page 694
2408	SQLCARD	*SQLCARD* on page 702
240B	SQLCINRD	*SQLCINRD* on page 705
240E	SQLRSLRD	*SQLRSLRD* on page 713
240F	RDB	*RDB* on page 571
2410	FRCFIXROW	*FRCFIXROW* on page 370
2411	SQLDARD	*SQLDARD* on page 707
2412	SQLDTA	*SQLDTA* on page 708
2413	SQLDTARD	*SQLDTARD* on page 709

Code Point (Hex)	Term Name	Reference
2414	SQLSTT	*SQLSTT* on page 714
2417	LMTBLKPRC	*LMTBLKPRC* on page 400
2418	FIXROWPRC	*FIXROWPRC* on page 365
2419	SQLSTTVRB	*SQLSTTVRB* on page 716
241A	QRYDSC	*QRYDSC* on page 560
241B	QRYDTA	*QRYDTA* on page 561
241C	BNDEXSRQR	*BNDEXSRQR* on page 125
241D	BNDEXSOPT	*BNDEXSOPT* on page 124
241E	DFTPKG	*DFTPKG* on page 267
241F	PKGRPLALW	*PKGRPLALW* on page 514
2420	PKGRPLNA	*PKGRPLNA* on page 515
2421	BNDCHKONL	*BNDCHKONL* on page 120
2422	BNDNERALW	*BNDNERALW* on page 126
2423	BNDERRALW	*BNDERRALW* on page 122
2424	PKGATHRVK	*PKGATHRVK* on page 496
2425	PKGATHKP	*PKGATHKP* on page 492
2426	STRDELAP	*STRDELAP* on page 733
2427	STRDELDQ	*STRDELDQ* on page 734
2429	ISODATFMT	*ISODATFMT* on page 389
242A	USADATFMT	*USADATFMT* on page 886
242B	EURDATFMT	*EURDATFMT* on page 321
242C	JISDATFMT	*JISDATFMT* on page 396
242E	ISOTIMFMT	*ISOTIMFMT* on page 395
242F	USATIMFMT	*USATIMFMT* on page 887
2430	EURTIMFMT	*EURTIMFMT* on page 322
2431	JISTIMFMT	*JISTIMFMT* on page 397
2432	CSTSYSDFT	*CSTSYSDFT* on page 244
2433	CSTBITS	*CSTBITS* on page 241
2434	CSTSBCS	*CSTSBCS* on page 243
2435	CSTMBCS	*CSTMBCS* on page 242
2436	STTSCCCLS	*STTSCCCLS* on page 743
2437	STTASMEUI	*STTASMEUI* on page 740
2438	RDBRLSCMM	*RDBRLSCMM* on page 601
2439	RDBRLSCNV	*RDBRLSCNV* on page 602
243A	EXPNON	*EXPNON* on page 345

Code Point (Hex)	**Term Name**	**Reference**
243B	EXPALL	*EXPALL* on page 344
243C	DECDELPRD	*DECDELPRD* on page 260
243D	DECDELCMA	*DECDELCMA* on page 259
243E	SQLOBJNAM	*SQLOBJNAM* on page 712
2441	ISOLVLCHG	*ISOLVLCHG* on page 391
2442	ISOLVLCS	*ISOLVLCS* on page 392
2443	ISOLVLALL	*ISOLVLALL* on page 390
2444	ISOLVLRR	*ISOLVLRR* on page 394
2445	ISOLVLNC	*ISOLVLNC* on page 393
244C	SRVLCNT	*SRVLCNT* on page 722
244D	SRVLSRV	*SRVLSRV* on page 723
244E	SRVLST	*SRVLST* on page 725
244F	SRVPRTY	*SRVPRTY* on page 730

Index