ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

REMOTE DATABASE ACCESS SERVICE AND PROTOCOL

TR/30

December 1985

ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

REMOTE DATABASE ACCESS SERVICE AND PROTOCOL

TR/30

December 1985

Brief History

Work on an ISO Reference Model for Open Systems Interconnection started in 1977 with the formation of ISO/TC97/SC16 and the development of a Reference Model, now ISO 7498.

In ECMA, specifications have been developed for a series of standards addressing various levels of the ISO model. Work on the Remote Data Access Service and Protocol specification started in 1983 and was preceded by ECMA standards for Transport Protocol (ECMA-72), Session Protocol (ECMA-75), Presentation Protocols (ECMA-84 and ECMA-86) and Virtual File Protocol (ECMA-85).

This ECMA Technical Report is one of a set of specifications for Open Systems Interconnection, many of which are ECMA Standards. These standards are intended to facilitate homogeneous interworking between heterogeneous information processing systems. This specification is within the framework for the coordination of standards for Open Systems Interconnection which is defined in ISO 7498.

This specification is based on the practical experience of ECMA member companies world-wide, and on the results of their active participation in the current work of ISO and national standardization bodies in Europe and the USA. It represents a pragmatic and widely based consensus.

A particular emphasis of this specificationis to specify the homogeneous externally visible and verifiable characteristics needed for interconnection compatibility, while avoiding unnecessary constraints upon and changes to the heterogeneous internal design and implementation of the information processing systems to be implemented.

In the interest of rapid and effective standardization this specification is oriented towards well understood needs and is consistent with emerging database standards being created by ANSI X3H2, and being progressed by ISO TC97 SC21. It is intended to be capable of modular extension to cover future needs and to exploit developments in technology.

The principal characteristics of this Service and Protocol are that:

- it offers efficient remote access to databases,

- it supports a distributed database system.

This specification provides a general framework for remote access to databases of many types, with specific encodings for access using the Relational Model.

The report was adopted by the General Assembly of ECMA as ECMA/TR30 on December 12th 1985.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

1. GENERAL

1.1 SCOPE

This Technical Report ECMA-DB is a specification for a Remote Database Access Service and Protocol. It:

- defines a database model (see Section 2);
- defines the operations on the database model as abstract interactions between two users of the communications service, one of which is acting on behalf of an application program whilst the other is interfacing to a process that controls data transfers to and from the database;
- defines the protocol to support the above service and its mapping to the underlying presentation service;

This specification is targetted at database systems that support the Relational Model. However, it is envisaged that other database systems will support relational interfaces and that subsets or supersets of the protocol may be used with other database systems.

The RDA Service and Protocol are for the Application Layer of Open Systems Interconnection.

1.2 REFERENCES

ISO 7489 Data Processing - Open Systems Interconnection - Basic Reference Model

ISO DP 8822 IPS - OSI - Connection Oriented Presentation Service Definition

Draft Proposed American National Standard - Database Language - SQL

ISO DP 8824 IPS - OSI - Specification of Abstract Syntax Notation One (ASN.1).

ISO DP 8825 IPS - OSI - Basic Encoding Rules for Abstract Syntax Notation One.

1.3 GENERAL DESCRIPTION

This Specification defines a communication protocol for the interchange of control information and data between a database user, the Client, and a database service provider, the Server.

The implementor at the Client end-system is required to provide program interfaces to the RDA Service and to map this service interface to the RDA Protocol.

The implementor at the Server end system is required to provide a database management system which also supports the RDA Protocol. For definitional purposes (and to make use of the ANSI X3H2 Database Language SQL specification) the Server is envisaged as comprising three components:

- the Database Management System;
- the RDA communication facility;
- the Client/Server program.

The Client/Server program acts as an intermediary between the RDA protocol machine and the DBMS, translating messages and data into calls on the Database Control System.

The facilities that are the subject of this Specification may be grouped into:

- DBMS schema facilities;
- DBMS data manipulation facilities;
- DBMS recovery facilities;
- RDA facilities.

The RDA protocol depends upon underlying OSI presentation, session, and transport facilites.

1.3.1 DBMS Schema Facilities

These facilities consist of the Data Description subset and the Macro Definition subset.

Whilst some form of schema definition is a logical necessity there is no requirement to provide the facilities to users of the RDA Service. Also, it would be possible to have macros defined using some local interface instead of the RDA facility.

No facilities for manipulating the Schema or accessing its content are defined in this version of the specification.

1.3.2 Data Manipulation Facilities

These facilities consist of the data manipulation statements: DECLARE CURSOR, CREATE TEMPORARY TABLE, OPEN, CLOSE, FETCH, INSERT, UPDATE, SELECT, TEST, INVOKE, DELETE. No subsetting of these facilities is proposed, except for access control purposes.

1.3.3 DBMS Recovery Facilities

Four classes of recovery facilities are proposed:

- Class 0: No on-line recovery;
- Class 1: Minimum recovery;
- Class 2: One stage Commit and Rollback for transaction back out;
- Class 3: Two stage Secure and Commit for distributed processing.

The recovery facilities for distributed processing require an extension to the DL SQL facilities. The current specification supports Class 2 only.

1.3.4 RDA Facilities

The RDA facilities are structured at two levels. The first level merely reflects the structuring of the DBMS capability. The second level is concerned with the protocol and facilities for efficient exploitation of the communication services.

The following facilities are mutually independent:

- Use of macros;
- Single Statements;
- Statement Grouping;
- Bulk Read facility;
- Bulk Write facility;
- Commitment Control.

Within the bulk transfer services there is the option of checkpointing to enable transfers to be restarted at intermediate points after a disruption of the transfer.

The same levels of commitment control apply as for DBMS recovery.

1.3.5 Underlying Presentation Service Facilities

Support of the Commitment Control facilities and of the Bulk Transfer checkpointing facilities requires use of the ISO Session Basic Synchronized Service with the Duplex Functional Unit.

Meaningful transfer of information requires agreement between the Presentation Layer entities at each end-system of the encoding to be used. Implementors will state which encodings are possible.

1.4 DEFINITIONS

Database terms are introduced in section 2.

Open Systems Interconnection terms are defined in Appendices A and C. Terminology is consistent with International Standard 7598.

1.5 ACRONYMS

CCR	Commitment, Concurrency and Recovery
DBCS	Database Control System
DBMS	Database Management System
DDBMS	Distributed DBMS
DL SQL	Database Language SQL
IPS	Information Processing Systems
PCI	Protocol Control Information
OSI	Open Systems Interconnection
RDA	Remote Database Access
RDAP	Remote Database Access Protocol

Many other acronyms are introduced within the document as abbreviations for machines, states, and messages. The following tables of these abbreviations may be referenced:

E.2	Protocol Machines
Table E/1	State Codes
Table E/2	Failure States
Table E/3	Message Suffices
Table E/4	Protocol Messages

2. DATABASE CONCEPTS

2.1 INTRODUCTION

This section describes the characteristics of a database that are assumed by the model and identifies specifically those characteristics that are visible to remote users of the database. The Database Management System (DBMS) consists of all the compilers, utilities and database access software necessary to support the creation, management and use of the database. The term Database Control System (DBCS) specifically refers to the run-time component providing database access and manipulation facilities to application programs.

This section is expository and does not form part of the RDA Specification.

2.2 THE DATABASE AND DISTRIBUTED DATABASE MODELS

2.2.1 General Principles

A database is a co-ordinated body of data managed by a software entity termed a Database Management System (DBMS). The DBMS maintains the database in permanent storage. It controls and facilitates the storage and retrieval of data in the database.

The logical structure of the database is defined by a database description known as a Schema. This defines the names and characteristics of all the data elements that may be stored, the inter-relationships among data elements, and the constraints on the values they may take. Details of the mapping of data to storage and performance requirements are generally defined in the Internal Schema or Storage Schema so that the logical capabilities that are visible to programmers can be separated from the performance concerns which are the responsibility of a data administrator.

Access to the database by application programs may be via a procedural interface or by extensions to a programming language which we can imagine being compiled into code that invokes the same procedural interface. Since different application programs require access to different subsets of the database the data available at the procedural interface for any single connection is defined in a separate data definition, the Subschema or External Schema. The procedural interface specification defines the data manipulation functions available and their effects on the database.

Specifications of the data structure descriptions and the data manipulation functions have been developed by ANSI X3H2. The semantics of the functions described in this specification and their effects upon the database are defined in the DL SQL specifications, which will be the subject of an ISO Standard in due course.

The current ANSI DL SQL does not include a Subschema definition language. However, the Subschema required by this protocol may be identical with the DL SQL schema so that there is no conflict. Also, the Subschema invoked in RDA may be a relational database description which an implementor has provided for non-relational databases.

2.2.2 Remote Access to a Database

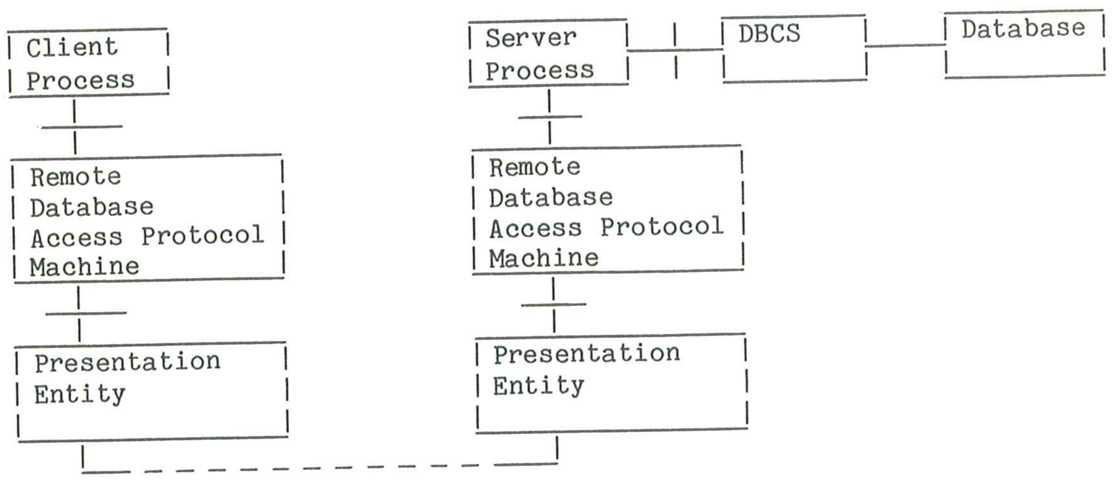
Client Process
Server Process
DBCS
Database
Remote Database Access Protocol Machine
Remote Database Access Protocol Machine
Presentation Entity
Presentation Entity

Fig. 1 - Structure of Remote Database Access Service

The Client process is the execution of an application program or Query Language Processor that has a data processing job to do. The RDA Protocol Machines are software components that handle the communications on behalf of the Client process and the Server process. At the database end the Server process is translating the protocol messages into Data Manipulation Procedural Interface calls and parameters and transmitting the results back using the communications service provided by the RDA Protocol Machine.

Note that, when the client process is a query processor, the work of analysing and executing the query is split between the front end, possibly in a workstation, and the back end in the server. The RDA provides independence between these components so that a user may use the same front end to access several different databases, and a single database may be shared by a population of users with different workstations and styles of man machine interface.

The diagram does not show the structure of the Client process. However it is envisaged that the service interface on the Client side will generally be driven by a component of a DBMS (or Distributed DBMS) so that the user interface for remote access to data is similar to the interface used for access to local data.

2.2.3 A Distributed Database or Multi-database System

A distributed database is a co-ordinated body of data that is partitioned into separated databases, each managed by its own DBMS. Co-ordination across the components is managed by a distributed DBMS (DDBMS) which is itself distributed.

It is possible for a Client process (or application program) to access the distributed database without being knowledgeable about the location of the data elements. Figure 2 shows a Client process calling a DDBMS component in its own end-system, which communicates with peer entities (DDBMS components) in other end-systems via the DDBMS connections.

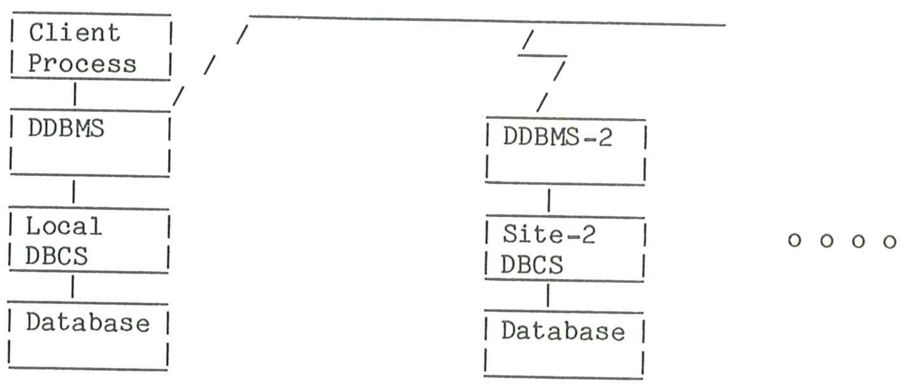

Fig. 2 - Distributed Database Scheme

Each connection between DDBMS components is a separate RDA Service connection. The lower layer connections will handle many transactions simultaneously, but the protocol defined in this specification is concerned with the interaction between the site with the Client process and one other site. Figure 3 shows the structure of a single connection.

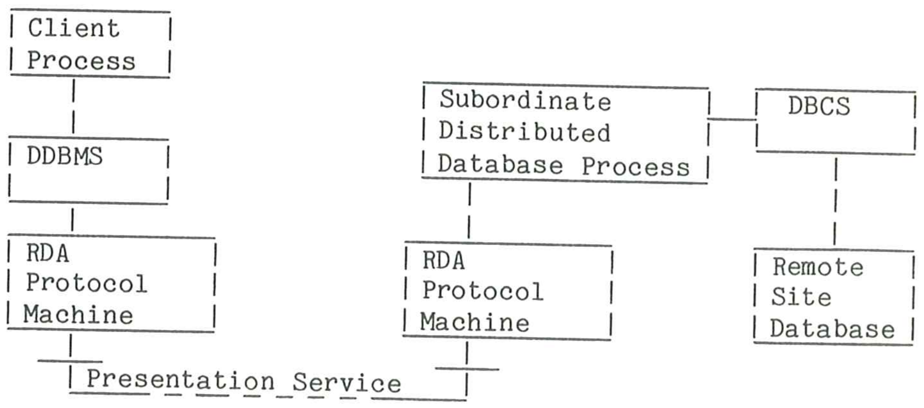

Fig. 3 - Distributed Database Single Connection

2.3 DISTRIBUTED DATABASE ATTRIBUTES

Each database has a database description (Schema) which describes the structure of the database and the allowable data values. It also contains authorization identifiers and privileges which constrain users' access to data.

The database is identified within an open system by the name of the application service and the database-name:

<database-name>	::= <database-service-name> <local-database-name>
<database-service-name>	::= global name necessary to establish connection to the Database Server supporting the remote database
<local-database-name>	::= name that must be passed to the DDBMS to enable it to identify the target database. The name must be unique amongst the database names known to the DDBMS.

The database terms used in this specification are defined hereafter. They are wholly consistent with the ANSI X3H2 Database Language SQL.

2.3.1 Data Types

A data type is a set of representable values.

A value is a null value or a non-null value.

The null value is a special value that is comparable with any value, but is distinct from all non-null values.

A non-null value is a character string or a number. A character string and a number are not comparable values.

The relationship between database values and transferred values is specified in 2.5.

2.3.1.1 Character Strings

A character string is a sequence of characters. The length of a character string is the number of characters in the sequence.

All character strings are comparable.

2.3.1.2 Numbers

A number is either an exact numeric value or an approximate numeric value. All numbers are comparable.

An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant decimal digits. A zero scale indicates that the number is an integer. For scale N, the exact numeric value is the integer value of the significant digits multiplied by 10 to the power -N.

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a single numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant digits in the mantissa.

2.3.2 Columns

A multi-set is an unordered collection of objects that are not necessarily distinct.

A column is a multi-set of values that may vary over time. All the values of the same column are of the same data type and are values in the same table. A value of a column is the smallest unit of data that can be selected from a table and the smallest unit of data that can be updated.

A column has a description and an ordinality within a table. The description of a column includes its data type, and an indication of whether the column is constrained to contain only non-null values. The description of a character string column specifies its length attribute. The description of an approximate numeric column specifies the precision of its numbers. The description of an exact numeric column specifies the precision and scale of its numbers.

2.3.3 Rows and Tables

A row is a non-empty sequence of values in a table. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

A table is a multi-set of rows. Every row of the same table has the same cardinality and contains a value of every column of that table. The nth value in every row of a table is the value of the nth column of that table.

A table has a description. The description includes a description of each of its columns.

A base table is a table that has a persistent storage representation and description. The description of a base table includes its name.

A derived table is an ephemeral table derived from one or more base tables during the execution of a <statement>.

A viewed table is a derived table that has a persistent description. The description of a viewed table includes its name.

A temporary table is a table whose storage representation and description persists only for the duration of the association between a Client and a Server. Temporary tables are not provided in ANSI X3H2 DL SQL.

2.3.4 Integrity Constraints

Integrity constraints define the valid states of the database by constraining the values that may be stored in the base tables of the database. A constraint can be associated with a single base table or with multiple base tables.

Integrity constraints are checked on execution of each statement that attempts to change the database. If the base tables associated with the integrity constraints do not satisfy the integrity constraints, the statement has no effect and the SQLCODE parameter is set to a negative value.

The ANSI DL SQL specification supports two types of constraint. A UNIQUE constraint specifies that no two rows of a table are allowed to have the same values for a specified column or columns. A NOT NULL constraint specifies that a given column is not allowed to have null values.

2.3.5 Authorisation Identifiers

An authorisation identifier is a string that designates a set of privilege descriptors. An authorisation identifier is applicable to the execution of every <statement>. For brevity, the term "authorisation ID" is used to refer to the applicable authorisation identifier of a <statement>. The method of establishing the authorisation ID is dependent on the implementation.

2.3.6 Privilege Descriptors

A privilege descriptor is a persistent object used by the implementation to enforce constraints on operations. If the operation specified in a <statement> is a constrained operation, the privilege(s) to perform that operation must be defined by the privilege descriptors designated by the authorisation identifier.

For brevity, the phrase "applicable privileges" is used to refer to the privileges defined by the privilege descriptors designated by the authorisation identifier.

2.4 ACTIVITY ATTRIBUTES

A Client process must establish an association with a database Server process before any manipulation functions can be carried out. The state of this association can be described, in part, by a set of activity attributes. These attributes may be classified as association attributes and data manipulation attributes.

2.4.1 Association Attributes

These are attributes that are established on the first connection within the association and remain constant until the connection terminates.

2.4.1.1 Identity of Initiator

This value is provided by the Client when the association is established. It is used by the Server to determine the Authorisation Identifier for the Client.

2.4.1.2 Account Code

This code identifies the account to which costs incurred on behalf of the client are to be charged.

2.4.1.3 Authorisation Identifiers

These are values known to the database management system. Each base table name and view name is associated with the authorisation identifier under which it was created. A Client may access tables that have the same Authorisation Identifer. For access to other tables the Client may need to use the appropriate Authorisation Identifier in order to specify an unambiguous reference. The attempted access will only be successful if the access is subject to a privilege granted by an authorised granter to his own Authorisation Identifier.

2.4.2 Data Manipulation Attributes

These are attributes that are created and deleted by the data manipulation functions invoked through the protocol.

2.4.2.1 Temporary Tables

A temporary table is a table which is created by the <create temporary table> statement. The values it contains are local to the creating process and may be updated by the process with no effect on other tables in the database. The cursor to a temporary table remains positioned after a <commit statement>. The table is deleted when the connection terminates.

2.4.2.2 Cursor

A cursor is specified by a <declare cursor> statement. An open cursor designates a table and a position relative to the rows of that table.

2.4.2.3 Macros

A macro is a named sequence of data manipulation functions that can be invoked by a single command. Macros may be permanent or temporary. Temporary macros are a property of a given association and cease to exist when the association terminates.

2.4.2.4 Transactions

A transaction is a logically complete unit of processing as determined by the application process. Only one transaction may be executed at any one time by a process, but the Server may be processing many transactions concurrently on behalf of different processes.

It is the Server's responsibility to guarantee transaction serialisability. That is, whenever a set of transactions is processed with overlapping existence times there must be some sequence of the transactions that would give exactly the same effect with no overlap between transactions. This specification does not dictate the mechanism by which serialisability is to be achieved.

2.5 DATA TRANSFER FORMATS

In this specification an abstract transfer syntax language, the ISO Abstract Syntax Notation One (ASN.1), is used to define the data content of messages exchanged between the Application Layer Entities. ASN1 is compatible with CCITT Recommendation X.409.

There is an encoding of this Abstract Syntax which determines the actual representation of data values. One encoding is defined in ISO DP 8825 (see refs). Other encodings are possible, defining alternative representations. The choice of representation may be negotiated between the communicating entities at connect time using the facilities of the Presentation Layer. ASN.1 is used to define both the Protocol Control Information and the user data.

In the RDA specifications the representation of data values is an implementor choice, but it is constrained by the set of values to be represented, which is defined in the DL SQL by the (data type) clause.

The data types that are specifiable in ASN.1 differ from the DL SQL data types in one important respect. ASN.1 allows the definition of Constructor types, which are named types whose values consist of choices or aggregations of other values.

The correspondence between the DL SQL types and the ASN.1 types is defined in the following table.

DL SQL Type	ASN.1 Type
CHARACTER [<length>]	STRING
FIXED <precision> [<scale>]	FIXED
NUMERIC <precision> [<scale>]	FIXED
INTEGER	INTEGER
FLOAT <precision>	FLOAT precision base
REAL	FLOAT precision base
DOUBLE PRECISION	FLOAT precision base

Note that ASN.1 requires that precision and scale be stated explicitly. However the representation as concrete syntax, the encoding of actual data, carries the length and precision but not the scale.

Floating point values will be encoded and sent with their implementor-defined precision. It is possible that the value received will not be identical with the value sent, but the difference between the values will be within the limit of the specified precision.

3. SERVICE

3.1 SERVICE OVERVIEW

This section provides an overview to the structure of a Remote Data Access Service and defines the service elements provided. The purpose of the facility is to enable a Client process to access and manipulate data which is at a remote location under the control of a database Server.

3.1.1 Roles of Partners

Before any work can be carried out the Client process must establish an association with the Server. An association is a relationship between the Client process and a responding Server process which survives until it is terminated in an orderly fashion. In particular it can survive interruptions in the underlying communication service or in the remote database service.

The communication service provides a connection between the Client process and the Server process. This supports a real-time dialogue which is essentially driven by the Client process which sends data manipulation or data management requests to the Server process. Hence the dialogue is asymmetrical.

However, there are phases within an RDA connection when data is being transferred in bulk. During these phases many data messages may be sent without individual acknowledgement. While bulk data transfer is in progress, temporary leadership is assumed by the sender of data, while the receiver acts as a slave, that is it can only accept data or report abnormal conditions. When bulk transfer is complete, the roles revert to Client and Server.

In this specification the term Association is used to define the relationship that exists between the Client and the Server from the time when it is first established to the time when it is terminated. An Association cannot be terminated without the agreement of both parties. It therefore survives any interruptions in the service however they may be caused. The term Connection is used to describe a live relationship that exists between the Client, the Server and the Communications Service. In the event of any sort of failure the connection is terminated and a new one must be established to continue with the Association.

3.1.2 Dynamic Structuring of an RDA Connection

A single RDA connection supports the database functions of a single Client process operating on one database. The Client process may operate on other databases, either within the same system, or in other end systems using other RDA connections. Also, other Client processes may operate on the same database concurrently using different RDA connections. Over a single RDA connection, operations on the database are executed in sequence in the order of submission.

The work performed on the database over an RDA connection can be modelled as a set of states, such that service elements, which imply transitions between states, are only valid when the connection is in an appropriate state.

Figure 4 shows the principal states and state transitions over an RDA connection. This model is very much simplified, in particular it does not show the effects of interruption and reconnection of the communication service. The states are described briefly in 3.1.2.1 to 3.1.2.5 below.

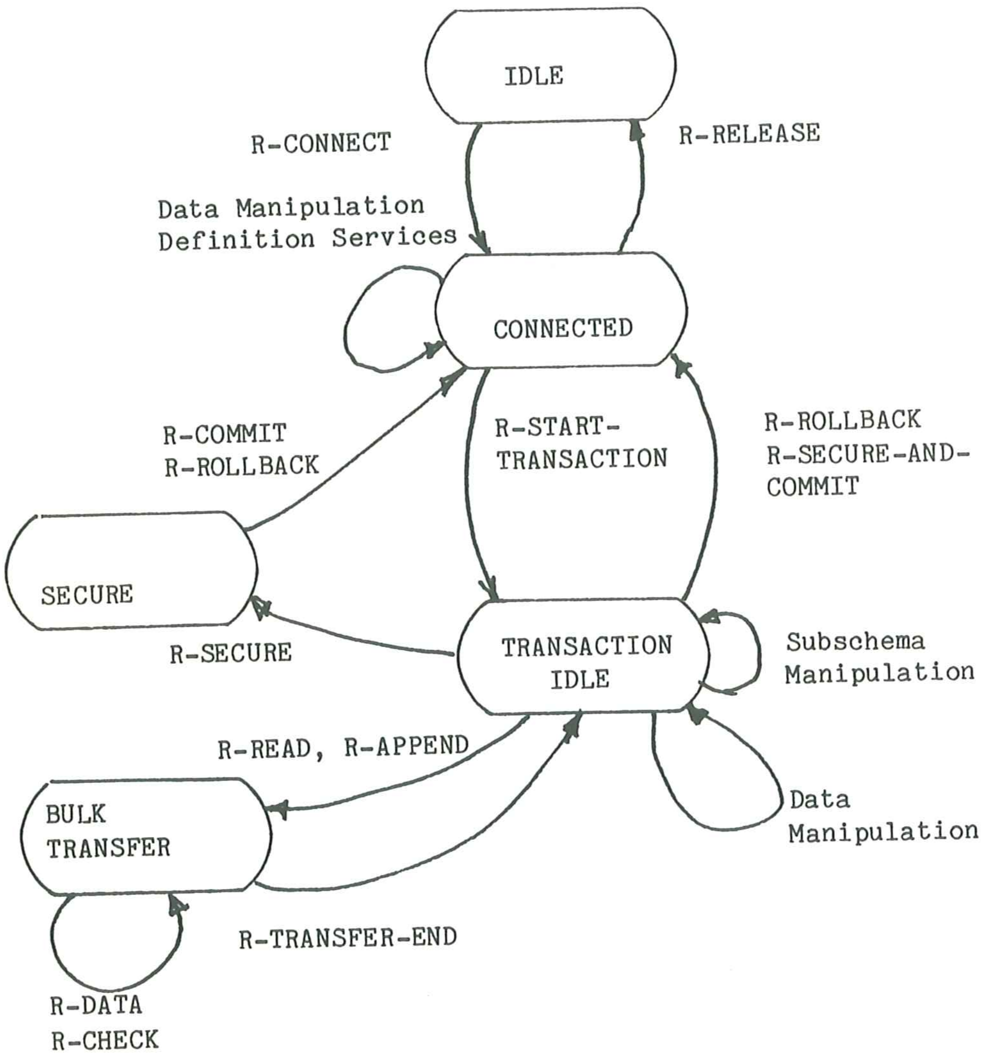

Fig. 4 - State Transition of the Service for Normal Operation

3.1.2.1 <u>Idle State</u>

In this state no connection exists. The only allowable service element is R-CONNECT.

3.1.2.2 Connected

In this state connection has been established. Service elements that affect association attributes are allowed but do not change the state. Either subschema manipulations or database transactions may be started by the R-START-TRANSACTION service elements.

3.1.2.3 Transaction Idle

In this state a transaction is in progress. Subschema manipulation functions are allowed. The transaction shall be terminated by R-ROLLBACK, R-SECURE, or R-SECURE-AND-COMMIT service elements.

3.1.2.4 Bulk Transfer

In this state the service elements R-DATA and R-CHECK leave the connection in Bulk Transfer state. When the transfer is complete the RDA connection returns to the Transaction Idle state.

3.1.2.5 Secure

In this state the processing for a transaction is complete and the Server is awaiting confirmation from the Client that other processes involved in the transaction have been completed. R-COMMIT or R-ROLLBACK service elements return the RDA connection to the Connected state.

3.1.3 Connection Services

The connection service elements provide for the establishment and release of the RDA connection. Release may be orderly or abrupt. Reconnection of a broken connection is also provided.

At connection establishment, there is negotiation of the particular class of service to be used and of any special convention that may be agreed.

Connection termination is normally requested by the Client process when all work is complete. However, in an emergency the connection may be abnormally terminated by either user at any point in time. It may also be accidentally lost, in which case both users are informed.

Reconnection is provided so that an interrupted bulk transfer can be resumed, and so that a transaction left in a secure state can be completed. Reconnection may be attempted by either the Client or Server. Reconnection by the Server is useful if the presentation service has been lost when the Server is in a secure state or when the Server has requested suspension within a bulk transfer operation.

3.1.4 Subschema Management Services

Subschema management service elements allow the user to create, alter and delete information stored in the Subschema description. Note that a change in the Subschema may necessitate a change to the underlying schema. The mapping of the Subschema to the schema is implementor defined. This information concerns database tables, database views, user privileges and macros.

For the present, no facilities to access or alter the subschema data defining the structure of tables or views are provided.

Users, or users groups, are identified by authority identifiers. Permission to perform given data manipulation functions on data in the database is granted explicitly to authority identifiers. Such permissions are recorded in the Subschema and modified by R-GRANT and R-REVOKE service elements.

Macro handling functions declare and delete macro definitions. A macro defines a sequence of data manipulation functions that can be invoked in a single service element. The R-CREATE-MACRO service element is used to declare a macro and R-DROP-MACRO removes a macro from the Subschema.

3.1.5 Data Manipulation Definition Services

The data manipulation definition services allow the Client to condition the association outside a transaction context. They allow temporary tables, cursors and temporary macros to be defined which exist until they are deleted or until the association is terminated. If these were created within a transaction they would disappear when the transaction terminated.

3.1.6 Transactions

The start of a transaction is signalled by the R-START-TRANSACTION service element when the association is in the connected state. Within a transaction, service elements are provided for the orderly completion or cancellation of the changes made to the database. Rollback of the transaction may be caused by either the Client or the Server in the event of a transaction or database error. If the transaction affects this database only it may be completed by the single R-SECURE-AND-COMMIT service element. If more than one database is affected the Client will need a confirmed R-SECURE from each Server before the transaction can be committed.

If the transaction fails when in the secure state it will be necessary to reconnect so that the Client can either commit or rollback the transaction. Reconnection may be attempted by either the Client or Server.

3.1.7 Data Manipulation Functions

The service element in this category carries data manipulation commands that operate on complete tables or on single rows of tables. In some cases the operation is notional, in the sense that a new table may be defined which is not actually materialised until or unless it is required by another function.

All data manipulation commands take effect within a transaction context and their effects are not visible to other database users until and unless the transaction is committed.

The responses to commands are dependent on the type of the command. Both the command and the response may carry data.

The commands available are listed below.

Command	Description
DECLARE CURSOR	Names a cursor
CREATE	Create Temporary Table
OPEN	Open cursor
CLOSE	Close cursor
FETCH	Get next row and move cursor
INSERT	Create new rows in a table
UPDATE	Update Rows of a table
DELETE	Erase rows of a table
INVOKE	Execute macro
SELECT	Specify and return a table of one row
TEST	Test for error response

Note that all commands except CREATE, INVOKE and TEST are defined in the DL SQL specification.

3.1.8 Bulk Data Transfer

The bulk data transfer service elements provide for the transfer of multiple rows, uninterrupted by data manipulation commands. The data flow from sender to receiver may be for the purpose of transferring a complete table to the Client or for appending a table in the Client Process to a table in the database.

Facilities are provided for:

- orderly termination of the transfer by the Sender with acknowledgement by the Receiver,
- abnormal termination by either user,
- checkpointing and checkpoint acknowledgement,
- restart of a transfer from a negotiated checkpoint position. This may be requested by either user and may follow a short interruption and reconnection of the communication service.

3.2 SERVICE DEFINITION

This service carries DL SQL statements and data to the Server process and carries responses and data from the Server to the Client process. The communication service need not be cognizant of the meaning of the statements except in so far as they affect the state of the protocol machine. The aspects of the activity that need specific definition are:

- Connection Management;
- Transaction Management;
- Grouping of Statements;
- Bulk Data Transfer.

3.2.1 List of Services

Table 1 lists all the service elements of the RDA Service. For each service element it specifies the type (see Appendix C), the user who can initiate it (CL: Client, SE: Server, SN: Sender, RC: Receiver) and its purpose.

CONNECTION

Service	Type	Init.	Description
R-CONNECT	2	CL	Establish RDA Connection
R-RELEASE	2	CL	Orderly release of RDA Connection
R-DISCONNECT	1	CL,SE	Abrupt release of RDA Connection
R-ABORT	3	-	Loss of Presentation Connection
R-RECONNECT	2	CL,SE	Establishment of interrupted Association

Table 3.1 Remote Data Access Service Elements (start)

TRANSACTION MANAGEMENT

Service	Type	Init.	Description
R-START-TRANSACTION	2	CL	Start Transaction
R-SECURE	2	CL	Intention to commit
R-COMMIT	2	CL	Transaction committed
R-SECURE-AND-COMMIT	2	CL	Request to commit Transaction
R-ROLLBACK	2	CL	Rollback Transaction
R-ROLLBACK-PLEASE	1	SE	Server cannot complete transaction

GROUPING AND DATABASE ACTIONS

Service	Type	Init.	Description
R-BEGIN-GROUP	2	CL	Indicate start of group
R-END-GROUP	2	CL	Terminate group
R-DL-DO	2	CL	Requests a DL SQL Action
R-STILL-PROCESSING	1	SV	Indicates delayed response
R-DEFINE-MACRO	2	CL	Defines a Macro
R-DROP-MACRO	2	CL	Drops a Macro

BULK TRANSFER

Service	Type	Init.	Description
R-APPEND-TABLE	2	CL	Start bulk write
R-READ-TABLE	2	CL	Start bulk read
R-DATA	1	SN	Send data
R-END-TRANSFER	2	CL	Terminate Bulk Transfer
R-END-READ	1	SN	Notify end of data
R-CHECK	2	SN	Establish checkpoint

Table 3.1 Remote Data Access Service Elements (continuation)

R-RESTART	2	SN,RC	Interrupt and negotiate restart
R-CANCEL	2	SN,RC	Cancel a transfer

Table 3.1 Remote Data Access Service Elements (end)

3.2.2 Notation

The notation used in the service specification is defined in Appendix C.

3.3 CONNECTION MANAGEMENT

This clause describes the service primitives for connection and disconnection of a Client process to a service being provided by a Server. The Server is assumed to have a Presentation Service Address which will enable the local presentation service to send a connection request to the correct entity in the network. This clause does not address the problem of network resources management, nor how the Client gets to know the correct current Presentation Service Address.

A connection only lasts for the time between the acceptance of a connection request and its orderly or abrupt (e.g. through communication failure or Server failure) termination. An association is a durable relationship which may span a number of connections. An association may be suspended and restarted, or it may be restarted after a break.

This clause describes the service elements necessary to support associations and for checking their availability when the initial connection is established.

3.3.1 Service Definition

The following service elements and primitives are described in this clause:

R-CONNECT	request, indication, response, confirm
R-DISCONNECT	request, indication
R-ABORT	indication
R-RELEASE	request, indication, response, confirm
P-RECONNECT	request, indication, response, confirm

3.3.2 The R-CONNECT Service Element

- Purpose: To request establishment of an association and connection between the initiating (Client) service user and the Server.

- Structure: Confirmed, type 2, RC.

- Parameters:

	Request	Indic.	Response	Confirm
Called Address	D	U		
Calling Address	D	U		
Association ID	D	U		
Responding Address			D	U
Class of R-Service	D	U	D	U
Quality of R-Service	D	U	D	U
Identity of Client	D	U		
Current Account	D	U		
User Data	D	U		
Diagnostic			D	U

Called Address

The Called Address identifies the database Server to which a connection is to be established. By implication, it also identifies the presentation, session and transport addresses supporting the remote entity.

Calling Address

The Calling Address is the R-service access point address from which the connection is established. The remote entity will receive an address which identifies the calling entity, and includes the underlying presentation, session and transport addresses.

Association Identifier

The Association Identifier, together with the Calling Address, uniquely identifies this association.

Responding Address

The Responding Address is the R-service access point address to be used in re-establishing an association after failure. It is not necessarily textually identical to the Called Address.

Class of R-Service

The Class of R-Service parameter conveys the service subset to be used. The values indicate the set of optional service features required in this association.

Quality of R-Service

The Quality of Service parameter conveys the quality of service requested for the association, and the quality of service offered by the responder.

Identity of Client

The Identity of Client parameter identifies the calling user. In the RDA Service it used by the Server to determine the Authorisation Identifier.

Current Account

The Current Account parameter identifies the account to which costs incurred in this association are to be charged.

User Data

The User Data parameter allows for the transfer of application service dependent information between users. It might, for example, be used for the authentication and management aspects of the application association.

In this application service the parameter carries an optional Subschema name.

This data has a size limit.

Diagnostic

This parameter contains a severity and reason code which indicates whether the connection has been established.

3.3.3 The R-DISCONNECT Service Element

- Purpose: To request immediate disconnection of a connection. It may be used either by either the Client or by the Server process. The service is disruptive and information may be lost.

- Structure: Non-confirmed, type 1, RI.

- Parameters:

	Request	Indication
Suspend	D	U
User Data	D	U

Suspend

The Suspend parameter indicates whether the association or merely the connection is being terminated. If true, and the quality of service supports associations, then R-RECONNECT may be successful.

User Data

May be used to supply a reason for disconnection, to suggest a suitable re-connect time, etc. This data is application dependent.

3.3.4 The R-ABORT Service Element

- Purpose: To inform application entities of the collapse of the presentation service, possibly caused by failure of the underlying transport service.
- Structure: Indication only, type 3, II.
- Parameters:

	Indication
Diagnostic	U

Diagnostic

This parameter contains a severity and reason code which indicates whether or not the connection has been established.

3.3.5 The R-RELEASE Service Element

- Purpose: To request suspension or completion of an association by agreement of both processes and without loss of information.
- Structure: Confirmed, type 2, RC.
- Parameters:

	Request	Indic.	Response	Confirm
Diagnostic			D	U
Suspend	D	U	D	U
User Data	D	U	D	U

Diagnostic

This parameter contains a severity and reason code which indicates whether or not the connection has been terminated.

Suspend

To Suspend parameter indicates whether the association or merely the connection is being terminated. If true, and the quality of service supports associations, then R-RECONNECT may be successful.

User Data

May be used to supply a reason for disconnection, to suggest a suitable re-connect time, etc. This data is application dependent.

3.3.6 The R-RECONNECT Service Element

- Purpose: To re-establish an association with a new connection after voluntary or involuntary disconnection of a previous connection. R-RECONNECT will not succeed unless the initial connection established availability of the facility in the quality of service negotiation. If both the Client and Server issue an R-RECONNECT, the Client will not deliver the request from the Server and the Server will respond to the Client.

- Structure: Confirmed, type 2, RC.

- Parameters:

	Request	Indic.	Response	Confirm
Called Address	D	U		
Calling Address	D	U		
Association Id.	D	U		
Responding Address			D	U
Class of R-Service	D	U	D	U
Quality of R-Service	D	U	D	U
Identity of Client	D	U		
Current Account	D	U		
User Data	D	U		
Diagnostic			D	U

Parameters have the same meaning as in R-CONNECT (3.3.2). The reconnecting entity shall re-establish the connection with the same values for the association attributes as those negotiated for the previous connection.

3.4 TRANSACTION MANAGEMENT

This clause is concerned with Commitment, Concurrency and Recovery within a distributed system. The CCR service elements govern the initiation of a recoverable unit of work and its orderly completion.

The discussion on CCR below applies to a wide class of distributed processing applications. This class includes distributed databases.

A Commitment unit is a unit of processing that makes a consistent change to the shared environment, principally the data resources that are required by other processes, since these persist after the process is complete.

The essential requirement for Concurrency is that the effect of processes (Commitment units) which overlap in time should be the same as it would have been if they had been executed one after the other. This is termed serializability of Commitment units. The resource providers are required to ensure that Commitment units never interfere with one another. The requirement can be stated more formally as follows. Whenever a set of Commitment units is processed there must exist at least one ordering of the Commitment units such that the effect of processing the Commitment units serially in the sequence determined by the ordering is identical to that of processing the set.

Recovery facilities must be provided by the resource managers. They must ensure that once a commitment is completed, it stays completed. it Also, if a Commitment unit does not successfully complete then must have no effect on the "visible" environment, i.e. on the data resources available to this or other Client processes. This specification is not concerned with the mechanisms for handling Recovery, but only with the inter-communication between the driving processes and the independently recoverable sub-systems.

Commitment units may be local and simple, or at the other extreme they may be distributed and complex. In order to handle the distributed case correctly it is necessary to give each Commitment unit a network-wide identifier. This consists of the identity of the master process that initiates the Commitment unit and an identifier allocated by that process.

Figure 5 shows a distributed Commitment unit in which the master process spawns sub-processes, some of which spawn other sub-processes.

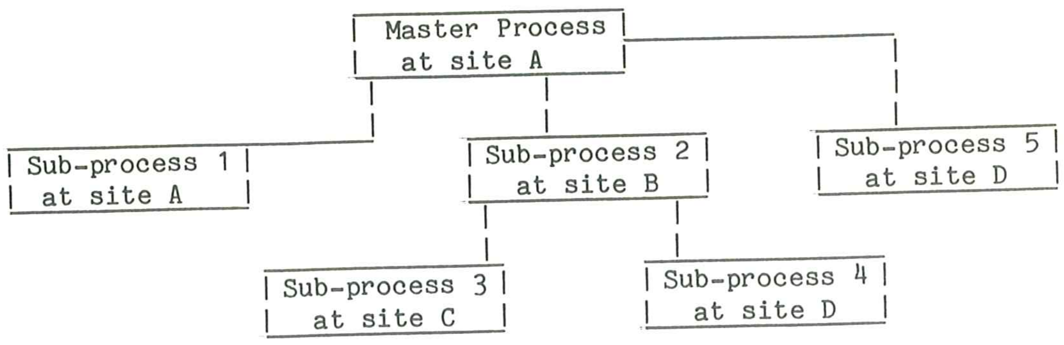

Fig. 5 - A Distributed Commitment Unit.

The sub-processes are not necessarily performing simple independent tasks. Each could involve multiple interactions with the process or sub-process that created them.

Note that sub-processes 4 and 5 are both at site D and may therefore clash over resource requirements. Since both processes carry the same Commitment unit identifier one process can release resources to the other before the commit point without losing the integrity of the Commitment unit.

In this application the Master Process is always a Client and a terminal (leaf) sub-process is always a Server. However an intermediate sub-process such as sub-process 2 may have both roles but with respect to different connections.

The example given is of the most exacting requirement. Many applications will require less complete facilities. The classes of CCR service that are envisaged are described below.

- Class 0 - No Recovery

  In the event of failure the state of the remote process is unknown. On establishment of a new connection the Client process will have to check and tidy up the environment before re-starting. For example, an interrupted file transfer may result in an incomplete file which would have to be deleted.

- Class 1 - Minimum Recovery

  In the event of failure the current activity is either completed or rolled back. There is an understanding that certain operations or sequences of operations constitute integral work units or Commitment units.

- Class 2 - Commitment

  The definition of Commitment units is part of the application protocol. When a Commitment unit is complete this is indicated to the remote process, which will either confirm that the process has been completed or roll it back and request restart. This protocol is not suitable for use by more than one recoverable sub-process within a Commitment unit. Note that when the master process requests commitment it effectively relinquishes control over the fate of the Commitment unit at that point.

- Class 3 - Distributed Commitment

  This protocol involves negotiation between all the recoverable sub-systems in the Commitment unit before the process is comitted. Each sub-system is requested to secure its updates in such a way that it can guarantee successful completion of the next request from the Client which will either be to rollback the transaction or to complete it. Only when all sub-processes have confirmed this, is the unit ready to be committed.

  The data secured by sub-units must be able to survive local system failure or communications failure. It must be possible to re-establish failed associations in order to complete secure processing.

3.4.1 Service Specification

The following service elements and primitives are defined in this clause:

R-START-TRANSACTION	Class 2, 3
R-SECURE	Class 3
R-COMMIT	Class 3
R-SECURE-AND-COMMIT	Class 2, 3
R-ROLLBACK	Class 2, 3
R-ROLLBACK-PLEASE	Class 2, 3

These service elements effectively bracket the work requested by the driving process of a subordinate process.

If level 0 or level 1 CCR Quality of R-Service has been negotiated none of the above service elements are available. In this case all the service elements defined in this specification as available within a transaction (in the TRANSACTION IDLE state) are available after connection (in the CONNECTED state).

One stage commitment is available if level 2 or level 3 CCR Quality of R-Service has been negotiated. Two-stage commitment negotiation is only available when level 3 CCR Quality of R-Service has been negotiated.

Any application protocol may impose its own rules governing the states at which it is permissible to start and commit Commitment units. For example, in a File Access Service, it may not be permissible to open a file outside a Commitment unit and close it within it.

Also, it may be possible to support Commitment units but revert to a level 1 Recovery strategy if the processing involves a single recoverable sub-unit and the master process chooses not to invoke the CCR service elements.

In a distributed process when one process has many subordinates the sequence of commitment is such that no process may be committed until all others are secure. It is possible for the last sub-process to be committed using Secure-and-Commit, instead of two-stage commit. If this is done then control over the fate of the Commitment unit is effectively handed over to the last sub-ordinate process until the Secure-and-Commit response is received.

3.4.2 The R-START-TRANSACTION Service Element

- Purpose: To indicate the start of a transaction and request confirmation that the remote process has established a rollback point. A transaction may not be initiated until any previous transaction is complete.

- Structure: Confirmed, type 2, RC.

- Parameters:

	Request	Indic.	Response	Confirm
Commitment Unit ID	D	U		

Commitment Unit Identifier

Identifies this Commitment unit. It identifies the master process and has a unique serial number allocated by that process. In the event that the remote process invokes another process it must start a new transaction with that process using the same Commitment Unit Identifier before accessing any data resources.

3.4.3 The R-SECURE Service Element

- Purpose: To request the remote process to secure its processing so that the success of a subsequent R-COMMIT or R-ROLLBACK can be guaranteed. R-SECURE indicates that the driver process has completed the transaction and is preparing to commit it. When the master process has requested R-SECURE the only permitted request primitives are R-COMMIT and R-ROLLBACK. If the underlying connection is interrupted (e.g. by R-ABORT) the association must be re-established so that the Commitment unit can be terminated.

- Structure: Confirmed, Type 2, RC.
- Parameter:

	Request	Indic.	Response	Confirm
Diagnostic			D	U

Diagnostic

Indicates success or failure. If failure is indicated the remote process has been rolled back.

3.4.4 The COMMIT Service Element

- Purpose: To inform the remote process that the Commitment unit is complete. The remote process can relinquish its ability to rollback to the previous commit point and release any locked resources under its control.
- Structure: Confirmed, Type 2, RC.
- Parameters: None.

3.4.5 The R-SECURE-AND-COMMIT Service Element

- Purpose: To inform the remote process that the master process is secure and request the remote process to commit its processing. In a distributed Commitment unit this service element may be used only when the requestor is a master or a process executing an R-SECURE-AND-COMMIT and when all other sub-processes of the requesting process are secure.
- Structure: Confirmed, Type 2, RC.
- Parameter:

	Request	Indic.	Response	Confirm
Diagnostic			D	U

Diagnostic

Indicates success or failure. If failure is indicated the remote process has been rolled back.

3.4.6 The R-ROLLBACK Service Element

- Purpose: To abort the current transaction and revert to a no-transaction context. An R-ROLLBACK request may only be made after an R-START-TRANSACTION and before an R-COMMIT or R-SECURE-AND-COMMIT.

- Structure: Confirmed, Type 2, RC.

- Parameter:

	Request	Indic.
Diagnostic	D	U

Diagnostic

A warning is given if rollback is attempted when no transaction is in progress.

3.4.7 The R-ROLLBACK-PLEASE Service Element

- Purpose: To abort the current transaction within a Server sub-process and revert to a no-transaction context. R-ROLLBACK-PLEASE is used by the subordinate process when it finds that it cannot complete the transaction for any reason. The driving process must then rollback the subordinate process using R-ROLLBACK.

- Structure: Unconfirmed, Type 1, RI

- Parameter:

	Request	Indic.
Diagnostic	D	U

Diagnostic

The Diagnostic parameter indicates either that the whole transaction must be rolled back or that it is possible to restart this sub-process independently of other sub-processes.

3.5 GROUPING OF STATEMENTS

Within the context of an association and a transaction the normal unit of data manipulation between the client process and the server is the equivalent of a DL SQL statement. The statement is the smallest unit of work recognised by the DBCS and each statement either succeeds or fails. If it fails it has no effect on the database or on the association attributes such as cursors. A statement is manifested at the service interface by a service element.

Each statement sent to the Server elicits a response, but it is not necessary for the Client process to wait for the response before sending the next statement. However there are times when the application logic requires a response before the next statement can be determined and it is useful to indicate these times to the communication service. This is accomplished by the service elements R-BEGIN-GROUP and R-END-GROUP.

The Client's view is that the service elements (statements) following the R-BEGIN-GROUP are actioned when the R-END-GROUP is sent and the confirmations are received at this time. The underlying communications service may deliver requests at any time (in the correct sequence) but the R-END-GROUP forces delivery to the Server.

3.5.1 Service Definition

The following service elements and primitives are described in this clause:

R-BEGIN-GROUP
R-END-GROUP
R-DL-DO
R-STILL-PROCESSING

R-BEGIN-GROUP signals the start of a sequence of statements, each of which is carried by an R-DL-DO. The group is terminated by an R-END-GROUP.

3.5.2 The R-BEGIN-GROUP Service Element

- Purpose: To indicate the start of a sequence of statements which are to be responded to together.
- Structure: Unconfirmed, Type 1, RI.
- Parameters: None.

3.5.3 The R-END-GROUP Service Element

- Purpose: To indicate the end of the group of statements and to request the responses.
- Structure: Confirmed, Type 2, RC.
- Parameters: None.

3.5.4 The R-DL-DO Service Element

- Purpose: To request the remote database Server to execute a single DL SQL statement and to return a response. The effect of the statement depends on the type of statement.

- Structure: Confirmed, Type 2, RI.

- Parameters:

	Request	Indic.	Response	Confirm
Error Action	D	U		
Statement	D	U		
Status			D	U
User Data	D	U	D	U

Error Action

In the event that this statement does not have the expected result the whole group may be terminated, or the next statement may be obeyed regardless.

Statement

The DL SQL statement. The permissible statements are defined in 3.5.5 and 3.7.

Status

Contains the SQLCODE returned from the DBCS after execution of the statement.

User Data

Contains data for storing in the database or data returned from the database.

3.5.5 DL SQL Statements

In the DL SQL specifications there are statements for manipulating schema information and statements for manipulating data in the database. A statement is specified within a procedure, which is part of a module.

In the RDA Service there are also data description statements and data manipulation statements, but there are no module declarations. Statements may either be specified within macros and given an interface or they may be sent directly using the R-DL-DO service element.

The meaning of the DL SQL statements to the DBMS is fully defined in the DL SQL specification, with the exception of a small number of extensions that are defined in this specification. The permissible statements are as follows:

- Schema Manipulation

  CREATE TEMPORARY TABLE: Defines a new table in the schema

  For the present the CREATE TEMPORARY TABLE statement represents a DL SQL extension. The table created only exists for the duration of the association.

- Data Manipulation

DECLARE CURSOR	: Declares a cursor and associates it with a table
OPEN	: Activates a cursor
CLOSE	: Disables a cursor
FETCH	: Moves the cursor to the next row and retrieves its column values
INSERT	: Inserts a row in a table
UPDATE	: Changes column values in a row or rows
DELETE	: Deletes a row or rows from a table
SELECT	: Specifies a one-row table and retrieves values
TEST	: Checks for exceptions in a previous statement
INVOKE	: Invokes a macro

- Commitment Control

SECURE	: Prepares a transaction for commitment
SECURE AND COMMIT	:One phase transaction commitment
COMMIT WORK	: Terminates a transaction successfully
ROLLBACK WORK	: Terminates a transaction and cancels its effects.

In the RDA Service Commitment Control is supported by specific service elements (see 3.4).

In the DL SQL the statements are defined as setting values in a parameter list, this being a reference to the parameters of the enclosing procedure within the module. Within the R-DL-DO these statements identify a row of values which is returned in the response user data as a sequence of typed data values. Null values are represented by the NULL representation. Indicator parameters (to distinguish NULL from non null values) are not required in the RDA Service though they may appear in a local service interface.

3.5.6 The R-STILL-PROCESSING Service Element

- Purpose: To inform the Client that an R-DL-DO is being processed and that the response is likely to be some time in preparation.

- Structure: Unconfirmed, Type 1, RI.

- Parameters:

	Request	Indic.
Time	D	U

Time

The estimated time to complete the processing, in seconds.

3.6 BULK DATA TRANSFER

The Bulk Data Transfer service elements provide for the transfer of multiple rows, uninterrupted by data manipulation commands. The data flow from sender to receiver may be for the purpose of transferring a complete table to the Client or for appending a table in the Client process to a table in the database.

Facilities are provided for:

- orderly termination of the transfer by the sender with acknowledgement by the receiver;
- abnormal termination by either user;
- checkpointing and checkpoint acknowledgement;
- restart of a transfer from a negotiated checkpoint position. This may be requested by either user and may follow a short interruption and reconnection of the communication service.

Note that the successful transfer of a table to the Server requires that the table be consistent with the database and its schema. Data errors such as duplicate rows will result in the cancellation of the transfer by the Server and negotiation of a restart point before continuation.

3.6.1 The Service Description

The service elements and primitives for normal operation (i.e. not including error recovery) are shown in Fig.6 and Fig.7. These show the sequence of primitives for bulk transfers from the Client to the Server and from the Server to the Client process. Fig.6 shows the normal sequence of primitives for R-APPEND TABLE, and Fig.7 the normal sequence of primitives for R-READ-TABLE.

For the duration of a bulk read, control is transferred to the Server. It is returned to the Client by the exchange of R-END-TRANSFER messages.

The R-DATA Service repeats while there is still data to be transferred. The transfers may be interspersed by requests to establish checkpoints.

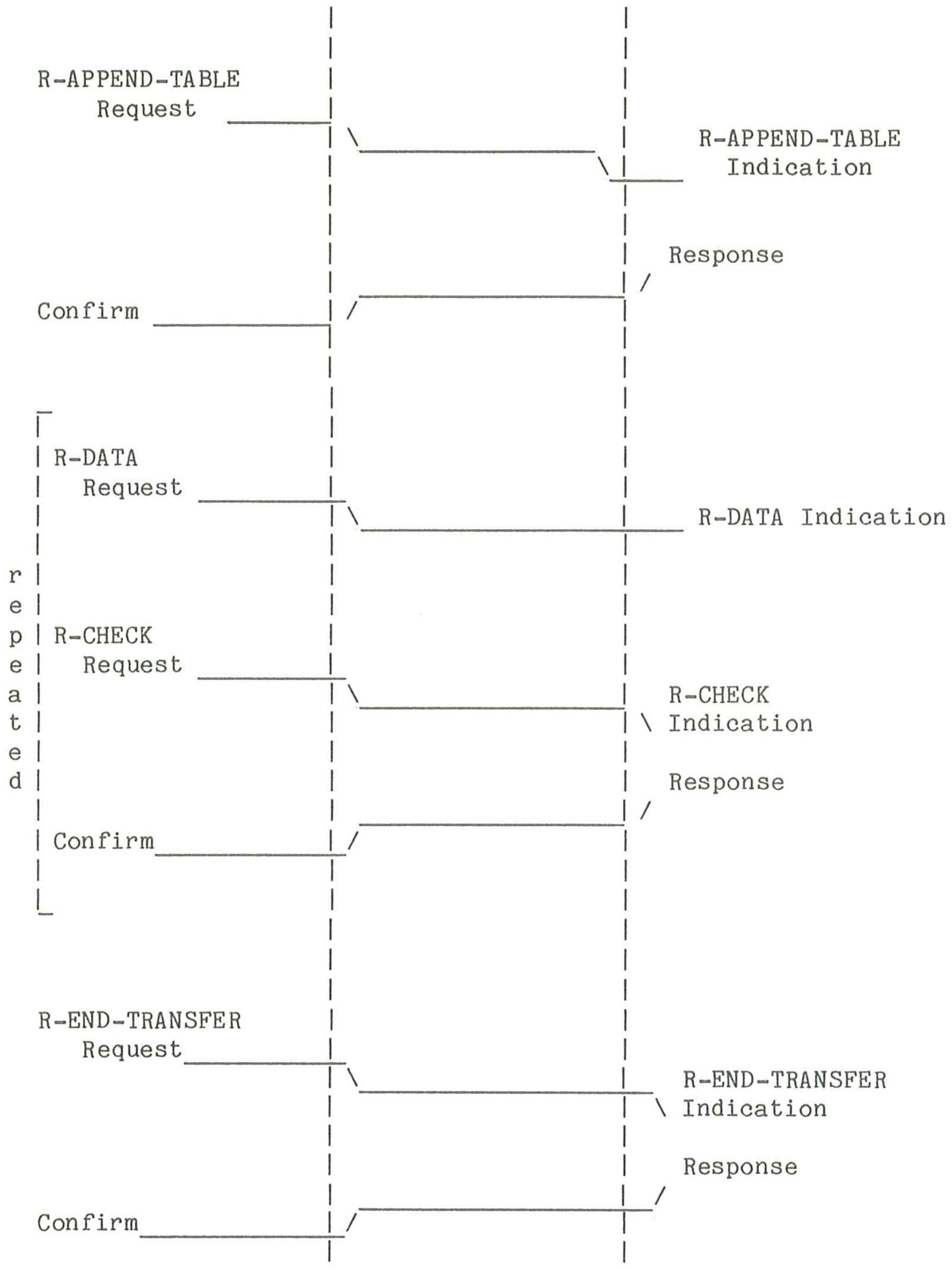

Fig. 6 Normal Sequence of Primitives
for R-APPEND-TABLE

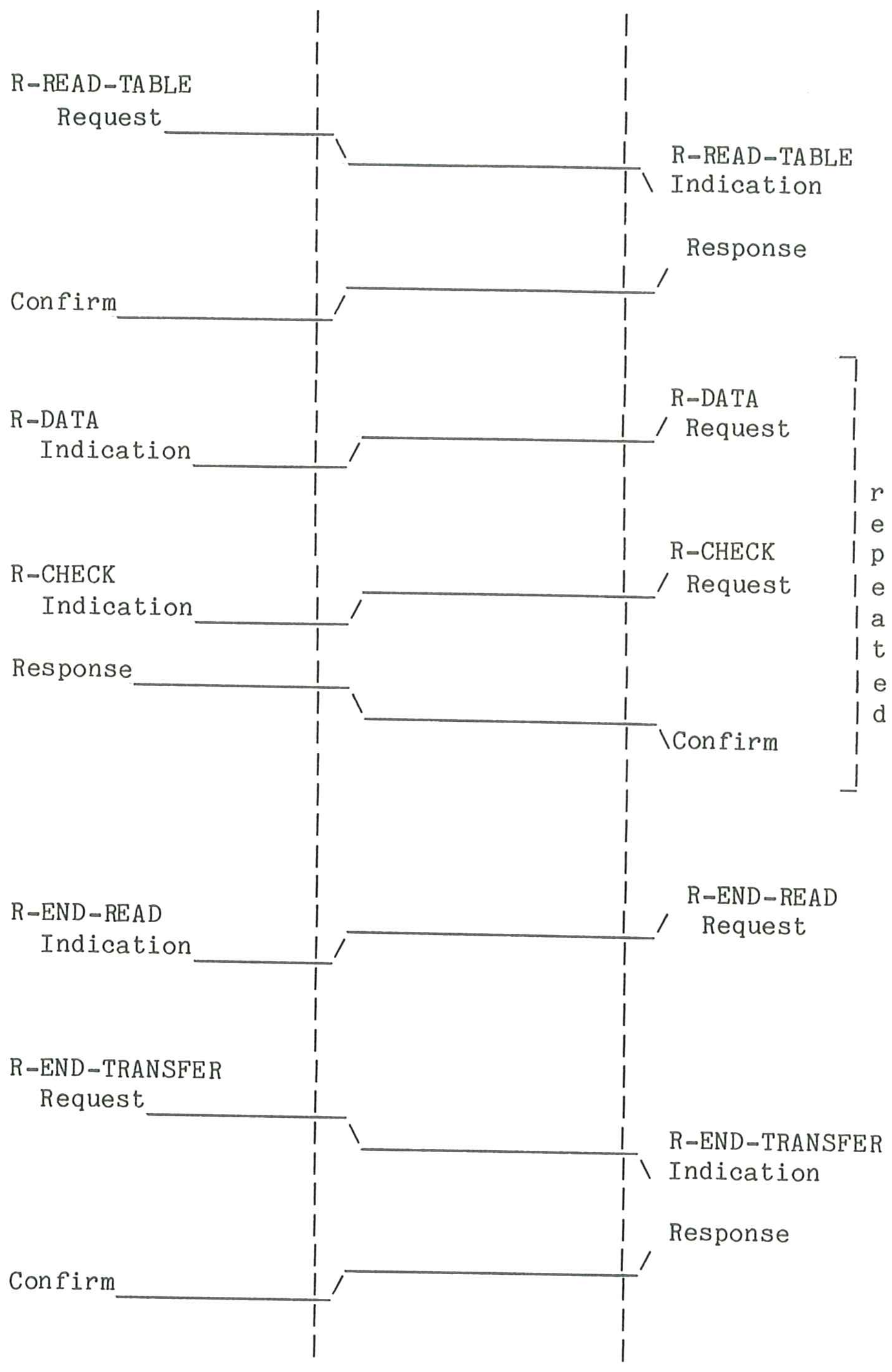

Fig. 7 Normal Sequence of Primitives for R-READ-TABLE

3.6.2 <u>The R-APPEND-TABLE Service Element</u>

- Purpose: To indicate to the Server that the Client is entering Bulk Transfer Mode.
- Structure: Confirmed, Type 2, RC.
- Parameters:

	Request	Indic.	Response	Confirm
Table Name	D	U		
Diagnostic			D	U

<u>Table Name</u>

Indicates a table in the database in which the new data will be stored.

<u>Diagnostic</u>

Indicates either that the transfer has been accepted or the reason why it has been rejected.

3.6.3 <u>The R-READ-TABLE Service Element</u>

- Purpose: These primitives specify a bulk transfer from the Server to the Client. Only one transfer may be in operation on the data object at any one time. The primitives signal a transfer of control from the Client to the Server. The R-READ-TABLE request may be rejected.
- Structure: Confirmed, Type 2, RC.
- Parameters:

	Request	Indic.	Response	Confirm
Table Name	D	U		
Select Expression	D	U		
Diagnostic			D	U

<u>Table Name</u>

Indicates a table in the database from which data will be read.

<u>Select Expression</u>

A DL SQL expression that specifies a derived table. This parameter is an alternative to the Table Name.

Diagnostic

Indicates either that the transfer has been accepted or the reason why it has been rejected.

3.6.4 The R-DATA Service Element

- Purpose: The transfer primitives transfer data from the sending process to the receiving process. The transfer of data may start following an R-APPEND-TABLE confirm or following receipt of an R-READ-TABLE indication. Data is transferred as sequences of rows of tables, using the underlying presentation facilities. Each row is a sequence of column values.

- Structure: Unconfirmed, Type 1, RI.

- Parameters:

	Request	Indic.
Database Data	D	U

Database Data

Each database data primitive carries an integral number of rows of a table. Each row consists of a sequence of column values.

3.6.5 The R-END-TRANSFER Service Element

- Purpose: The primitives for R-END-TRANSFER mark the completion of the transfer operation and return control to the Client process. The confirmation primitive indicates that no further error recovery actions will be requested for this transfer.

- Structure: Confirmed, Type 2, RC.

- Parameter:

	Request	Indic.	Response	Confirm
Diagnostic	D	U	D	U

Diagnostic

The Diagnostic parameter is supplied by the bulk transfer receiver. So after R-READ-TABLE the Diagnostic may be used in the request primitive; after R-APPEND-TABLE the Diagnostic may be used on the response primitive. The Diagnostic indicates success or carries warning information concerning the transfer.

3.6.6 The R-END-READ Service Element

- Purpose: To return control to the Client process after a bulk transfer from the Server to the Client.

- Structure: Unconfirmed, Type 1, RI

- Parameters: None.

3.6.7 Restart During Bulk Transfer

To protect themselves against failure in the Client process, Server or in the communications, the sender and receiver of data may establish checkpoints. In the event of failure the processes can re-establish the connection (if necessary), then agree a suitable point for restart of the transfer operation.

The commands included in this group are:

R-CHECK
R-RESTART
R-CANCEL

3.6.8 The R-CHECK Service Element

- Purpose: R-CHECK is used by the sender to propose a check point to the receiver. Checkpoints are sequentially numbered within the transfer operation. After receipt of a checkpoint indication and before sending the checkpoint response, the receiver must secure the data received up to that point and thereby establish a restart point. The sending process does not need to wait for the confirmation primitive, and the receiving process need not respond to every indication.

- Structure: Confirmed, Type 2, RC

- Parameter:

	Request	Indic.	Response	Confirm
Checkpoint Number	D	U	D	U

Checkpoint Number

Checkpoints are numbered sequentially by the sender within each transfer.

3.6.9 The R-RESTART Service Element

- Purpose: Restart primitives interrupt any activity in progress, with possible loss of undelivered indications or confirms. They negotiate a point at which data transfer can be restarted. If the receiver and sender both issue a restart request so that the requests cross, the receiving entity will not deliver the indication from the sender.

- Structure: Confirmed, Type 2, RC
- Parameters:

	Request	Indic.	Response	Confirm
Restart Point	D	U	D	U
Diagnostic			D	U

Restart Point

Indicates the checkpoint number at which data transfer is to restart. The bulk data Sender does not include the parameter. The Receiver determines the last suitable checkpoint.

Diagnostic

If no restart is possible, indicates the reason.

3.6.10 The R-CANCEL Service Element

- Purpose: Any one of the Client process, the Server process or the provider of the communications service may cancel a data transfer activity. The two users may then have different views of the state of the activity. The primitives interrupt any activity in progress (including R-RESTART) and any undelivered indications or confirms may be discarded. Further bulk transfer operations may be attempted. However, if the receiver is the Server and the Quality of Service includes recovery any effect of the transfer on the database will be reversed.
- Structure: Confirmed, Type 2, RC.
- Parameter:

	Request	Indic.	Response	Confirm
Diagnostic	D	U	D	U

Diagnostic

Indicates the reason for the R-CANCEL and whether restart may be attempted.

3.7 MACRO DECLARATIONS

A macro declaration consists of an interface spec and a body spec. The interface spec defines the name of the macro and specifies the parameters that are to be supplied when the macro is invoked and those that are returned as a result of its execution. The body defines the function of the macro and consists of a list of statements, possibly only one, that are obeyed in sequence each time the macro is invoked.

The macro declaration is equivalent to the following notional DL SQL syntax:

```
<macro spec>          ::= MACRO <scope> <interface spec>
                          <body spec>

<scope>               ::= TEMPORARY | PERMANENT

<interface spec>      ::=
           <macro name> [UPDATE <parameter spec>...]
                        [IN <parameter spec>...]
                        [OUT <parameter spec>...]

<parameter spec>      ::= <parameter name> <data type>

<body spec>           ::= <statement> ... END

<statement>           ::=
        <create temporary table>     | <close statement> |
        <cursor declaration>     | <delete statement: positioned> |
        <delete statement: searched>     | <fetch statement> |
        <insert statement>     | <invoke statement> | <open statement> |
        <select statement>     | <test statement> |
        <update statement: positioned> | <update statement: searched>
```

The fetch and insert statements return or require a single row of table data. The <data type> "row" designates a sequence of column values and matches a row of any table.

The TEST statement may only be specified within a macro or group. It serves to test the status of the previous statement and to conditionally exit from the macro or terminate the group.

3.7.1 <u>The R-DEFINE-MACRO Service Element</u>

- Purpose: To specify a new macro for the use of the Client process.

- Structure: Confirmed, Type 2, RC.

- Parameters:

	Request	Indic.	Response	Confirm
Scope	D	U		
Macro Name	D	U		
Macro Definition	D	U		
Extended Diagnostic			D	U

Scope

Specifies whether the macro is to be permanently stored in the database, and therefore available after the completion of this association, or temporary in which case it will be dropped when the association terminates.

Macro Name

Must be unique amongst the temporary macros or permanent macros.

Macro Definition

Defines the parameters on the user interface and the statements that form the macro body.

Extended Diagnostic

The extended diagnostic includes the standard diagnostic. A warning is returned if a temporary Macro Name clashes with a name already recorded in the Subschema for a permanent macro. The extended diagnostic also reports errors in the DL SQL statements.

3.7.2 The R-DROP-MACRO Service Element

- Purpose: To remove a macro from the Subschema.
- Structure: Confirmed, Type 2, RC.
- Parameters:

	Request	Indic.	Response	Confirm
Scope	D	U		
Macro Name	D	U		
Diagnostic			D	U

Scope

Specifies whether the macro is temporary or permanent.

Macro Name

Identifies the macro.

Diagnostic

Indicates that the macro has been deleted, warns that the macro does not exist or reports an error if the attempt to delete the macro is unsuccessful.

4 THE RDA PROTOCOL

4.1 PROTOCOL OVERVIEW

The Remote Data Access Protocol determines the structure and content of messages exchanged between the Client and Server processes. These messages are sent as a result of Request or Response Service Primitives and their arrival generally results in delivery of Indication or Confirmation Service Primitives to the receiving service user. The protocol specification determines the permissible content of the messages and the circumstances in which they may be dispatched. It also defines the expected behaviour of the receiving entities and that of the users of the services, i.e. the client and server processes.

4.1.1 Roles of RDAP Entities

The asymmetry of the RDA Service is reflected in the protocol. The two protocol machines play different and complementary roles, corresponding to the different roles played by their respective users: Client and Server outside a bulk transfer, Sender and Receiver when a bulk transfer is in progress.

4.1.2 Descriptive Model

The RDA Protocol is modelled as a set of elementary protocol structures between abstract protocol machines which exist at either end of the connection. A elementary protocol structure is a dialogue for the purpose of an indivisible operation. As such, it is totally successful or totally unsuccessful, never partially successful. It is composed of a request, issued by one RDAP entity and, for most but not all types of protocol structure, of a response issued by the other RDAP entity. Each request or response is a single protocol message.

A protocol message contains Protocol Control Information, PCI, (i.e. one or more parameters) and may contain user data.

The arrival of a message and the occurrence of a request or response service primitive are Events. An event causes the protocol machine to carry out a sequence of actions which leave the machine in a certain state. The actions performed by a machine depend on the state of the machine when the event occurs and on the type of event.

Dynamic execution of the RDA protocol results in an ordered sequence of protocol structures. To describe the protocol it is sufficient to describe each of the structures (or messages), plus any precedence relationships between structures (state transitions).

4.1.3 Use of the Presentation Service

The RDA Protocol makes use of the services provided by the underlying Presentation Service, which includes facilities that are provided by lower layers.

The following services are used:

P-CONNECT	Establish Presentation Connection
P-U-ABORT	Abrupt Termination by User
P-RELEASE	Orderly Termination
P-P-ABORT	Presentation Service Failure
P-TYPED-DATA	Send Application PCI and data
P-TOKEN-GIVE	Send tokens
P-TOKEN-PLEASE	Request tokens
P-SYNC-MINOR	Synchronise Minor
P-SYNC-MAJOR	Synchronise Major
P-RESYNCHRONISE	Resynchronise

It is assumed that negotiation of transfer formats will be carried out by the presentation layer entities without requiring communication between the RDAP entities. Negotiation between an RDAP Entity and a Presentation Entity in a single end-system is not part of the abstract service interface defined in this standard.

4.2 PROTOCOL DESCRIPTION

The text in this section is a plain English definition and explanation of the protocol. It should not be at variance with the Formal Description defined in Appendix E. Hopefully any ambiguities or omissions in the informal description can be resolved by reference to the formal definition.

The specification of the protocol is structured into groups of primitive dialogue elements according to their function. These are:

- Association Management - concerned with the establishment and release of connections and associations;

- Data Definition and Manipulation Management - concerned with schema manipulation, macros and temporary tables;

- Transaction Management and Data Manipulation - concerned with sharing, commitment, recovery from DBCS failure (e.g. deadlock), and Relational Database Language data manipulation operations;

- Bulk Data Transfer - concerned with the continuous transfer of complete tables, either from the Client to the database or vice versa;

- Grouping - facilities to group protocol elements to minimise the effects of transmission delays;

- Failure and Recovery within the RDA Service - concerned with abrupt disconnections and failures in the communications service, and with reconnection and resynchronisation after such failures.

The protocol defines the permissible behaviour of the protocol machines but not that of the remote DBCS. The protocol machine is not concerned with the detailed semantics of the RDL statements. Because of their complexity the permissible statements and transfer encodings are described separately.

Within each of the sections defining functional groupings the protocol definition defines the messages, the actions to be performed by the sending and receiving entities and the mapping of the message to the underlying Presentation Service.

The definition of each message follows the same form:

- Source (Client, Server, Sender or Receiver);
- Function;
- Message Content (parameters);
- Parameter Description;
- Sending Actions;
- Receiving Actions.

The mapping of RDAP messages to the service primitives of the Presentation Service, and the encodings for all Protocol Control Information are specified in Section 4.10. Abstract Syntax Notation One, ASN.1, is used to specify the Transfer Syntax. Appendix D describes the ASN.1 macros used in RDAP.

4.3 ASSOCIATION MANAGEMENT

This section defines the protocol elements concerned with the establishment and release of connections and associations between the Client and Server entities. The messages that support the service primitives are listed in the table below.

Function	Service Primitives	Messages	
Connection	R-CONNECT	CON	Connect
		CON-R	Connect Response
Disconnection	R-RELEASE	REL	Release
		REL-R	Release Response

These messages define establishment and controlled release of an association. Abrupt termination and recovery of an association are described in section 4.8.

4.3.1 The CON (Connect) Message

- Source: Client

- Function: To establish an association and set the initial values of association attributes.

- Contents:
  Called Address
  Calling Address
  Association ID
  Class of R-Service
  Quality of R-Service
  Identity of Client
  Current Account
  User data

- Notes on Contents:

  The meaning of the above parameters is defined in the Service Description.

- Sending Actions:

  The CON message is sent following receipt of a valid R-CONNECT Service Request and the establishment of a Presentation Connection with the appropriate Server.

- Receiving Actions:

  Receipt of a CON message causes the receiving protocol entity to deliver an R-CONNECT indication to the Server. The server is expected to check the access permissions of the client, and the possibility of the requested association attributes, before responding with an R-CONNECT response.

4.3.2 The CON-R (Connect Response) Message

- Source: Server

- Function:

  To respond to a CON message, either by confirming the establishment of the requested association or by refusing it with an appropriate diagnostic.

- Contents:
  Diagnostic
  Responding Address
  Association Identifier
  Class of R-Service
  Quality of R-Service
  User Data

- Notes on Contents:

  If the request is refused only the Diagnostic is mandatory. The Class of R-Service and the Quality of R-Service may be different to the values requested in the CON message.

- Sending Actions:

  The CON-R message is sent following receipt of a valid R-CONNECT Service Response. If the diagnostic indicates Success or Success with Warning, then a new Association has been established. The Server is in the Connected state.

  If the diagnostic indicates failure, the connection request has been refused. Any further messages received will be discarded, with the exception of CON.

- Receiving Actions:

  The content of the CON-R is delivered to the Client using an R-CONNECT Confirm primitive. If the diagnostic indicates failure any messages queued for sending will be discarded and the machine returns to the Idle state. If the Diagnostic indicates Success or Success with Warning the Association has been established and the state is Connected.

4.3.3 The REL Message

- Source: Client

- Function:

  To terminate the connection in an orderly manner and, optionally, to terminate the association.

- Contents:
  Suspend
  User Data

- Sending Actions:

  A valid R-RELEASE Request normally causes an REL release message.

- Receiving Actions:

  The receiving entity delivers an R-RELEASE indication and awaits the response. If 'suspend' is requested the server is expected to checkpoint the association attributes for re-use on a later connection, otherwise it may delete the temporary tables and macros belonging to the association.

4.3.4 The REL-R Message

- Source: Server

- Function: to respond to a Release request

- Contents:
  Diagnostic
  User data

- Notes on Contents:

  The diagnostic may indicate that a request to suspend has been refused.

- Sending Actions:

  The REL-R message is sent following a R-RELEASE Response. If the diagnostic indicates Success or Success with Warning the connection is terminated. If it indicates failure the connection remains open.

  If suspension is requested and accepted the connection is terminated but the association is suspended in Fail State F1, see 4.8.5.

- Receiving Actions:

  A R-RELEASE Confirm is delivered to the Client. If the Diagnostic indicated Success or Success with Warning the Presentation connection is terminated and no further request primitives can be accepted on this RDA connection. However if the association has been suspended, it will be possible to establish a new connection to resume it using R-RECONNECT.

  If the REL-R diagnostic indicates failure the Client is still in the connected state.

4.4 DATA DEFINITION AND MANIPULATION MANAGEMENT

This group of protocol elements covers the data definition and manipulation facilities allowed outside a transaction context. These include the definition of Macros for use within the transactions. Macros may be defined for this association only or for general use.

The messages defined in this group are shown in the table below.

Function	Service Primitives	Messages	
Define macro	R-DEFINE-MACRO	MDF	Define Macro
		MDF-R	Define Macro Response
Drop macro	R-DROP-MACRO	MDR	Drop Macro
		MDR-R	Drop Macro Response
Data Manipulation	R-DL-DO	RDL	DL SQL Functions
		RDL-R	RDL Response
	R-STILL-PROCESSING	SPR	Still Processing

Only the <create temporary table> and <cursor declaration> DL SQL Functions are allowed in this context.

4.4.1 The MDF (Define Macro)Message

- Source: Client
- Function: to define a new permanent or temporary macro
- Contents:
  Scope
  Macro name
  Macro definition
- Sending Actions:

  The MDF message is sent following a valid R-DEFINE-MACRO request.
- Receiving Actions:

  Receipt of a valid MDF message causes an R-DEFINE-MACRO indication to be delivered to the server. The server is expected to check the validity of the macro definition and, if correct, to record it in the subschema and make it available for use within the association.

4.4.2 The MDF-R (Define Macro Response) Message

- Source: Server
- Function: to confirm the acceptability of the newly defined macro or to provide a diagnostic
- Contents:
  Diagnostic
  Extended Diagnostic
- Sending Actions: The message results from an R-DEFINE-MACRO response primitive.

- Receiving Actions: The receiver sends an R-DEFINE-MACRO confirm primitive to the client.

4.4.3 The MDR (Drop Macro) Message

- Source: Client
- Function: to delete a permanent or temporary macro
- Contents:
  Scope
  Macro name
- Sending Actions: The MDR message is sent following a valid R-DROP-MACRO request.
- Receiving Actions:

  Receipt of a valid MDR message causes an R-DROP-MACRO indication to be delivered to the server. The server is expected to delete the referenced macro from the association and subschema. It will return a warning if the macro does not exist.

4.4.4 The MDR-R (Drop Macro Response) Message

- Source: Server
- Function: to confirm the deletion of a permanent or temporary macro or to provide a diagnostic.
- Contents:
  Diagnostic
- Sending Actions: The message results from an R-DROP-MACRO response primitive.
- Receiving Actions: The receiver delivers an R-DROP-MACRO confirm primitive to the client.

4.4.5 The RDL (DL SQL Functions) Message

- Source: Client
- Function: To carry one statement or one Macro invocation to the DBCS for execution.
- Contents:
  Error Action
  DL SQL Statement
  User Data

- Notes on Contents:

  In the protocol, all statement types are permissible. However the DBCS will reject statements that are not available to this user or are being used in the wrong context. In particular, statements that manipulate database data are not permissible outside a transaction when database sharing and recovery are being supported.

- Sending Actions:

  The RDL message is sent following a valid R-DL-DO Request.

- Receiving Actions:

  The receiver delivers an R-DL-DO Indication to the server. The server is expected to cause the DBCS to execute the statement and to return the results in an R-DL-DO Response.

4.4.6 <u>The RDL-R (RDL Response) Message</u>

- Source: Server

- Function: To confirm the execution of a DL SQL statement and to return the results.

- Contents:
  Status
  User Data

- Sending Actions: The message is sent following a valid R-DL-DO Response primitive.

- Receiving Actions: The receiver delivers an R-DL-DO Confirm to the Client.

4.4.7 <u>The SPR (Still Processing) Message</u>

- Source: Server.

- Function: To inform the Client that processing is continuing and may take an extended time period. Use of SPR is optional.

- Contents:
  Time

- Sending Actions:

  The message is sent by the Server on receipt of a valid R-STILL-PROCESSING Request.

- Receiving Actions:

  The Receiver delivers an R-STILL-PROCESSING indication to the Client

4.5 TRANSACTION MANAGEMENT AND DATA MANIPULATION

The transaction maanagement messages are concerned with the initiation and completion of transactions. A transaction may complete successfully or it may be cancelled by either party. The DBCS uses the rollback facilities to return the database to the rollback point established at the start of the transaction.

Transaction Management messages are only available when level 2 or level 3 CCR Quality of R-Service has been negotiated. If level 0 or level 1 CCR has been negotiated the data manipulation messages are available outside a transaction.

The messages in the transaction management and data manipulation groups are shown in the table below:

Function	Service Primitives	Messages	
Start Transaction	R-START-TRANSACTION	STR	Start Transaction
		STR-R	Start Transaction Response
Secure	R-SECURE	SEC	Secure
		SEC-R	Secure Response
Secure and Commit	R-SECURE-AND-COMMIT	SCM	Secure and Commit
		SCM-R	Secure and Commit Response
Commit	R-COMMIT	COM	Commit
		COM-R	Commit Response
Rollback	R-ROLLBACK	RBK	Rollback
		RBK-R	Rollback Response
	R-ROLLBACK-PLEASE	RBP	Rollback Please
Data Manipulation	R-DL-DO	RDL	DL SQL Functions
		RDL-R	RDL Response
	R-STILL-PROCESSING	SPR	Still Processing

Data Manipulation is included in this group and in the previous group because the service elements may be used both inside and outside a transaction. However when transactions are supported different sets of statements may be sent within the different contexts. The messages are described above in sections 4.4.5 and 4.4.6.

4.5.1 The STR (Start Transaction) Message

- Source: Client
- Function: to indicate the start of a transaction and require the DBCS to establish a rollback point.
- Contents:
  Transaction ID
- Sending Actions:

  The STR is sent following a valid R-START-TRANSACTION request from the Client process. Transactions may not be nested, so the primitive is only valid when the Client protocol machine is in the connected state.
- Receiving Actions:

  The receiver delivers an R-START-TRANSACTION indication to the server. The server is expected to establish a secure rollback point so that the association's view of the database state can be returned to this point if the transaction fails to complete.

4.5.2 The STR-R (Start Transaction Response) Message

- Source: Server
- Function: to confirm the establishment of a rollback point and the start of the new transaction.
- Contents:
  empty.
- Sending Actions: sent as a result of an R-START-TRANSACTION response primitive.
- Receiving Actions: The receiver delivers an R-START-TRANSACTION confirm to the Client.

4.5.3 The SEC (Secure) Message

- Source: Client
- Function: To indicate the end of processing for the current transaction and requests the server to prepare to commit it.
- Contents: none
- Sending Actions: A valid R-SECURE request primitive causes a SEC message to be sent.

- Receiving Actions:

  The receiver delivers an R-SECURE indication to the server. The server is expected to preserve the effects of the transaction in secure store prior to full commitment or transaction rollback.

4.5.4 The SEC-R (Secure Response) Message

- Source: Server

  Function: To inform the Client either that the server process (and any subordinate processes) have successfully secured the transaction or that they have failed to secure it and have rolled it back.

- Contents:
  Diagnostic
  Transaction ID

- Notes on Contents:

  The Secure request may be rejected with a Failure diagnostic, indicating that the server has successfully rolled back the transaction.

- Sending Actions:

  The message is sent following an R-SECURE response. If the diagnostic indicates success the server is in the secure state, awaiting transaction commitment or rollback. If the diagnostic indicates failure the server is in the connected state, i.e. it is no longer within a transaction.

  The server may repeat the R-SECURE Response primitive if it is in the Secure state, causing a repeat SEC-R to be sent. However, in the event of a clash between a SEC-R message and either an RBK or a COM message the SEC-R is discarded.

- Receiving Actions:

  Receipt of an SEC-R message causes an R-SECURE Confirm to be delivered to the Client. If the diagnostic indicates failure, the protocol machine is in the Connected state, outside a transaction context.

4.5.5 The SCM (Secure and Commit) Message

- Source: Client

- Function: To inform the server that this transation is complete and that the server can commit the transaction. In the event that no response is forthcoming from the server the message may be repeated.

- Contents:
  Transaction ID

- Sending Actions:

  The message is sent following a valid R-SECURE-AND-COMMIT request primitive from the client. If the message clashes with an RBP the client loses and the Rollback request is passed to the client.

- Receiving Actions:

  The SCM message causes the R-SECURE-AND-COMMIT indication to be delivered to the server. The server is expected to attempt to commit the transaction. If the attempt is unsuccessful the transaction must be rolled back. The R-SECURE-AND-COMMIT Response primitive must report the success or failure of the attempt.

  If an SCM message arrives referencing a transaction that has already completed, a copy of the SCM-R is sent back.

4.5.6 The SCM-R (Secure and Commit Response) Message

- Source: Server

- Function: to confirm the success or failure of the attempt to commit the transaction.

- Contents:
  Diagnostic

- Sending Actions:

  The message is sent following an R-SECURE-AND-COMMIT response. It completes the transaction and returns the server to the Connected state, outside any transaction context. Note that the server retains a memory of the disposition of the transaction.

- Receiving Actions:

  An R-SECURE-AND-COMMIT Confirm is delivered to the Client. The client is then in a Connected state, outside any transaction context.

4.5.7 The COM (Commit) Message

- Source: Client
- Function: to inform the Server process that the transaction is complete. The message may only be sent following receipt of SEC-R, i.e. when the process is secure.
- Contents: none
- Sending Actions: Sent following an R-COMMIT request.
- Receiving Actions:

  Receipt of a COM message causes an R-COMMIT indication to be delivered to the server. This completes the transaction.

  The server will release any resources, such as record locks, retained by the transaction.

4.5.8 The COM-R (Commit Response) Message

- Source: Client
- Function: to confirm the completion of the transaction.
- Contents: none.
- Sending Actions: A valid R-COMMIT Response causes this message to be sent.
- Receiving Actions: The receiver causes an R-COMMIT Confirm to be delivered to the client. This terminates the transaction and leaves the connection in the Connected state.

4.5.9 The RBK (Rollback) Message

- Source: Client
- Function: To abort the transaction by requesting that the server return the database to its state at the start of the transaction.
- Contents:
  empty.
- Sending Actions:

  The message is sent following an R-ROLLBACK Request. Any messages ahead of the RBK may be discarded. All messages received from the server are discarded until the request is acknowledged. If no RBK-R is forthcoming the request may be repeated.

- Receiving Actions:

  If this is the first RBK for this transaction the receiver causes an R-ROLLBACK indication to be delivered to the server. It may also discard any reply messages waiting to be dispatched. If an RBK has already been received this one may be discarded.

  The server is required to roll back the transaction and cause an R-ROLLBACK response.

4.5. 10 The RBK-R (Rollback Response) Message

- Source: Server

- Function: To confirm that the transaction has been rolled back.

- Contents:
  Diagnostic.

- Sending Actions:

  The RBK-R is sent following an R-ROLLBACK response primitive. The server is returned to the Connected state outside a transaction context.

- Receiving Actions:

  An R-ROLLBACK Confirm is delivered to the Client. The connection is then in the connected state outside a transaction context.

4.5. 11 The RBP (Rollback Please) Message

- Source: Server

- Function: to inform the Client that the transaction has been aborted by the Server.

- Contents:
  Diagnostic

- Notes on Contents:

  The diagnostic indicates whether restarting the transaction will inevitably repeat the failure or may possibly succeed.

- Sending Actions:

  The message is sent following an R-ROLLBACK-PLEASE Request. The server may delete any messages that are awaiting dispatch ahead of the RBP, and will ignore any messages other than RBK and disconnection messages, that would otherwise be allowed.

- Receiving Actions:

  An R-ROLLBACK-PLEASE Indication is delivered to the Client. The Client must issue an R-ROLLBACK Request in order to re-synchronise the service and clear any unwanted messages.

4.6 BULK DATA TRANSFER

This section describes the protocol elements concerned with transfer of bulk data. When bulk transfer mode is entered the roles of the protocol machines change from Client and Server to Sender and Receiver. However the change to the new roles does not happen simultaneously at each end.

Bulk data transfer normally occurs within a transaction. In this case the whole transfer may be rolled back, but the transfer must be terminated before rollback can be invoked.

The messages defined in this section are shown in the table below:

Function	Service Primitives	Messages	
Start Transfer	R-APPEND-TABLE	APT	Append Table
		APT-R	Append Table Response
	R-READ-TABLE	RDT	Read Table
		RDT-R	Read Table Response
Normal	R-END-READ	ERD	End Read
Termination	R-END-TRANSFER	ETR	End Transfer
		ETR-R	End Transfer Response
Sending data	R-DATA	DAT	Data
	R-CHECK	CHK	Check
		CHK-R	Check Response
Error Control	R-CANCEL	CAN	Cancel
		CAN-R	Cancel Response
	R-RESTART	RST	Restart
		RST-R	Restart Response

4.6.1 The APT (Append Table) Message

- Source: Client

- Function: to request entry into Bulk Transfer sending mode.

- Contents:
  Table Name

- Sending Actions: The message is sent following a valid R-APPEND-TABLE Request. The checkpoint counter is set to zero.

- Receiving Actions: An R-APPEND-TABLE Indication is delivered to the Server. The server is expected to respond with an R-APPEND-TABLE Response.

4.6.2 The APT-R (Append Table Response) Message

- Source: Server

- Function: To accept or reject an APT request.

- Contents:
  Diagnostic

- Sending Actions:

  The message follows an R-APPEND-TABLE Response primitive. If the diagnostic indicates Success, the Server becomes a Receiver and is in the Receiving state. If the diagnostic indicates failure the state reverts to that prior to the APT, i.e. Transaction Idle.

- Receiving Actions:

  The receiver delivers an R-APPEND-TABLE Confirm to the Client. If the diagnostic indicates Success the Client becomes the Sender and is in the Sending state. If the diagnostic indicates failure the state reverts to that prior to the APT, i.e. Transaction Idle.

4.6.3 The RDT (Read Table) Message

- Source: Client

- Function: to request entry into Bulk Transfer mode and specify the data to be transferred to the Client.

- Contents:
  Table Name
  Select Expression

- Sending Actions: The message is sent following a valid R-READ-TABLE Request. The checkpoint counter is set to zero.

- Receiving Actions: An R-READ-TABLE Indication is delivered to the Server. The server is expected to respond with an R-READ-TABLE Response.

4.6.4 The RDT-R (Read Table Response) Message

- Source: Server

- Function: To accept or reject an RDT request.

- Contents:
  Diagnostic

- Sending Actions:

  The message follows an R-READ-TABLE Response primitive. If the diagnostic indicates Success, the Server becomes a Sender and is in the Sending state. If the diagnostic indicates failure the state reverts to that prior to the RDT, i.e. Transaction Idle.

- Receiving Actions:

  The receiver delivers an R-READ-TABLE Confirm to the Client. If the diagnostic indicates Success the Client becomes the Receiver and is in the Receiving state. If the diagnostic indicates failure the state reverts to that prior to the RDT, i.e. Transaction Idle.

4.6.5 The ERD (End Read) Message

- Source: Sender, but only when the Sender is the Server.

- Function: To indicate that all data has been sent and to request termination of the transfer.

- Contents: none

- Sending Actions: Sent following an R-END-READ Request. The sender then waits for an ETR message to terminate the bulk transfer.

- Receiving Actions: The receiver delivers an R-END-READ indication to the Receiver Entity, which is also the Client. The client is expected to terminate the transfer with an R-END-TRANSFER request, but it may still acknowledge checkpoints, request cancellation or request restart.

4.6.6 <u>The ETR (End Transfer) Message</u>

- Source: Sender or Receiver, Client

- Function: To return from Bulk Transfer mode to normal Client/Server working.

- Contents:
  Diagnostic

- Notes on Contents:

  The diagnostic parameter is only used following a read, i.e. when the Client is also the Receiver.

- Sending Actions:

  The message is sent following an R-END-TRANSFER request. When the Client is the Sender this is used to indicate the end of data. If the Client is the Receiver it reverts to the Client machine at this point; otherwise it must remain in the Sender role because receipt of the data has not been confirmed.

- Receiving Actions:

  An R-END-TRANSFER Indication is delivered to the Server. The server is expected to reply with an R-END-TRANSFER Response. However if the server is also the receiver it should first secure the data received, and it may send checkpoint acknowledgements first, or request restart or cancel the transfer.

4.6.7 <u>The ETR-R (End Transfer Response) Message</u>

- Source: Server

- Function: To confirm the completion of Bulk Transfer and to revert to normal Client/Server working.

- Contents:
  Diagnostic

- Notes on Contents:

  The diagnostic parameter is only used following an append i.e. when the Server is also the Receiver.

- Sending Actions: The message follows an R-END-TRANSFER Response. It leaves the Server in the Transaction Idle State.

- Receiving Actions: An R-END-TRANSFER Confirm is delivered to the Client, which is now in the Transaction Idle state.

4.6.8 The DAT (Data) Message

- Source: Sender

- Function: To carry user data.

- Contents:
  User Data

- Sending Actions:

  Sent following an R-DATA Request primitive. Normally a sequence of DAT messages will be sent. The order of these, and any interspersed CHK messages is preserved by the RDAP and Presentation Services.

- Receiving Actions: An R-Data Indication is delivered to the Receiver.

4.6.9 The CHK (Check) Message

- Source: Sender

- Function: To mark a possible restart point in the data flow.

- Contents:
  Checkpoint Number

- Sending Actions:

  Sent following an R-CHECK request at the exact position in the data flow. The checkpoint number is provided by the sender. The protocol machine keeps a count of the number of unacknowledged checkpoints outstanding.

- Receiving Actions:

  An R-CHECK indication to the receiver. The receive is expected to mark the checkpoint and to acknowledge reciept of the check when it has been recorded in secure store. There is no requirement for immediate acknowledgement.

4.6.10 The CHK-R (Check Response) Message

- Source: Receiver

- Function: To acknowledge the securing of a checkpoint.

- Contents:
  Checkpoint Number

- Sending Actions: Sent, against the data flow, following an R-CHECK Response primitive.

- Receiving Actions: The CHK-R receiver decrements the count of outstanding checkpoints and delivers an R-CHECK Confirm to the Sender.

4.6.11 <u>The CAN (Cancel) Message</u>

- Source: Sender or Receiver

- Function: To abort the transfer, but to leave the association connected and within the transaction.

- Contents:
  Diagnostic

- Sending Actions:

  Sent as a result of an R-CANCEL Request. The protocol machine first deletes any messages waiting to be dispatched and any data, check or restart indications waiting to be delivered. If the request clashes with an incoming CAN message, then the Senders CAN wins. The protocol machine discards any further incoming messages, except connection termination messages, until the CAN is accepted.

- Receiving Actions:

  If the receiver is the Bulk Transfer Sender and it is waiting for a CAN-R the CAN is discarded. Otherwise any messages awaiting dispatch are discarded and an R-CANCEL indication is delivered.

4.6.12 <u>The CAN-R (Cancel Response) Message</u>

- Source: Sender or Receiver

- Function: to acknowledge cancellation of the transfer and return the connection to normal Client/Server working.

- Contents:
  Diagnostic

- Sending Actions:

  The CAN-R is sent following an R-CANCEL Response primitive. The protocol machine then reverts to the Transaction Idle state of the Client or Server machine. The next protocol exchange is expected to be initiated by the Client.

- Receiving Actions:

  An R-CANCEL Confirm is delivered to the recipient and the state reverts to Transaction Idle within the Client or Server machine.

4.6.13 The RST (Restart) Message

- Source: Sender or Receiver

- Function: To request restart from a checkpoint.

- Contents:
  Restart Point

- Notes on Contents:

  The restart point is the checkpoint number at which the transfer is to resume. The Sender may offer a restart point, but the Receiver may decide on an earlier one.

- Sending Actions:

  Sent following an R-RESTART Request. This interrupts the transfer of data and checkpointing messages and causes the protocol machine to disregard any that are waiting for dispatch or delivery and any received until the restart is acknowledged.

  If RST messages clash the Receiver message wins. If an RST message clashes with a CAN then the latter wins.

- Receiving Actions:

  The receiver delivers an R-RESTART Indication and awaits a response. Any data or check messages waiting for dispatch or delivery may be discarded.

4.6.14 The RST-R (Restart Response) Message

- Source: Sender or Receiver

- Function: To acknowledge the restart request.

- Contents:
  Restart Point
  Diagnostic

- Notes on Contents:

  The diagnostic may contain a warning, but refusal of a restart request should be accomplished by a CAN.

- Sending Actions:

  Sent following a R-RESTART Response. If the RST-R is from the Sender it may be followed immediately by DAT data messages. The count of outstanding checkpoints is set back to zero.

- Receiving Actions:

  An R-RESTART Confirm is delivered and the state reverts to Sending or Receiving. If the receiver is the Sender the count of outstanding checkpoints is set to zero.

4.7 GROUPING OF RDAP ELEMENTS

In general, the protocol is described as if the Client protocol machine waits for a response to each message it sends to the server before initiating the next protocol structure. This method of description simplifies the explanation of the protocol. However the RDAP permits the Client to group together a sequence of messages to mitigate the effect of end to end delays. Such a sequence is termed a Group and is delimited by Begin Group and End Group messages.

The grouping of protocol structures corresponds to the grouping of service structures described in 3.5. The effect of the grouping mechanism on the protocol is briefly described here. The same considerations apply to many messages and it would be confusing to repeat the description in every case.

- Effects at the Client

  First, it is necessary to verify that a request message can be legally part of the current group before issuing the message. Second, the state machine cannot immediately be progressed to the pending state: the new state that the sending of the message should cause is enqueued in a first-in-first-out queue. The enqueued states are successively dequeued as successful response messages are received and processed. The queue is purged when the end of group response is received.

- Effects at the Server

  The only effect at the server is when a failed response is issued; then, all incoming request messages are ignored by the Server until the next end of group message.

The messages that control grouping are:

Function	Service Primitives	Messages	
Grouping	R-BEGIN-GROUP	BGG	Begin Group
	R-END-GROUP	EGR	End Group
		EGR-R	End Group Response

4.7.1 The BGG (Begin Group) Message

- Source: Client
- Function: To mark the start of a group of protocol elements.
- Contents: none.
- Sending Actions:

  Sent following an R-BEGIN-GROUP Request. Groups may not be nested, so the protocol machine may not already be within a group. The sender sets up a queue for pending states for the protocol elements that will follow within the group. Confirmations from the Server need not be delivered to the Client until the R-END-GROUP request is received.

- Receiving Actions:

  The protocol machine delivers an R-BEGIN-GROUP indication to the Server and enters group mode. In this mode messages within the group are indicated to the server in the normal way, until any event occurs that invalidates the rest of the group. If this happens further incoming messages are discarded until the end of the group is reached.

4.7.2 The EGR (End Group) Message

- Source: Client
- Function: To indicate the end of a group.
- Contents: none.
- Sending Actions:

  Sent following an R-END-GROUP Request. This terminates the group and may be used to force delivery of the messages within the group.

- Receiving Actions:

  An R-END-GROUP Indication is delivered to the Server. This terminates the group mode. The server is expected to cause an R-END-GROUP Response.

4.7.3 The EGR-R (End Group Response) Message

- Source: Server

- Function: to acknowledge termination of the group and also to terminate a group early.

- Contents: none

- Sending Actions:

  Sent following an R-END-GROUP Response. This may occur ahead of the EGR if the group has to be terminated because of an error, or unfavourable database status. The Server will then ignore incoming messages until the EGR is received. This will not be delivered to the Server.

- Receiving Actions:

  If the EGR-R is received ahead of the R-END-GROUP Request the protocol machine will ignore any further Requests until the end of the group. An R-END-GROUP Confirm is delivered to the Client.

4.8 FAILURE AND RECOVERY WITHIN THE RDA SERVICE

It is posssible for a connection to be abruptly terminated by the Client, the Server or by the failure of the Presentation Service. After failure, provided that level 2 or level 3 CCR Quality of R-Service has been negotiated, it is possible to establish a new connection and continue with the association. Indeed, for application and database integrity reconnection is required whenever the disconnection leaves the database, or the Client, in a Secure state.

When the disconnection is abrupt, messages in transit may be lost and the two application entities may have different understandings of the state of the service. After reconnection it is necessary for them to synchronise their states before resumption of normal working.

The initiative for reconnection normally comes from either the Client or the Receiver, but the Server and Sender may also attempt reconnection. Consequently there may be clashes and confusion, since failure may occur when the application processes are changing roles.

The messages involved in failure and recovery are shown below:

Function	Service Primitives	Messages	
User Abort	R-DISCONNECT	DIS	Disconnect
Provider Abort	P-ABORT	PAB	Presentation Abort
Reconnect	R-RECONNECT	RCN	Reconnect
		RCN-R	Reconnect Response

This section first describes the messages, then continues with an explanation of the failure and resumption states.

4.8.1 The DIS (Disconnect) Message

- Source: Client, Server, Sender or Receiver
- Function: to unilaterally close down the connection
- Contents:
  Suspend
  User data
- Notes on Contents:

  Suspend indicates whether an attempt to reconnect may be successful.

  The user data field is of limited size. It may be used to carry a reason for the disconnection, such as protocol error.
- Sending Actions:

  The messsage follows an R-DISCONNECT Request. The protocol machine discards any messages waiting to be sent or delivered, apart from a DIS or PAB. If either of these is present it takes priority.
- Receiving Actions:

  On receipt of a DIS the receiving entity discards all messages waiting to be sent or delivered, and terminates the Presentation Connection.

4.8.2 The PAB (Presentation Abort) Message

- Source: Presentation Service.
- Function: to indicate the loss of the Presentation Connection.
- Contents: none.
- Receiving Actions:

  All messages awaiting dispatch or delivery are discarded and the Service User is informed by delivery of an R-ABORT Indication.

4.8.3 The RCN (Reconnect) Message

- Source: Client, Sender, Server or Receiver.

- Function: To request establishment of a new connection which is a continuation of a previous Association.

- Contents:
  Called Address
  Calling Address
  Association ID
  Class of R-Service
  Quality of R-Service
  Identity of Client
  Current Account
  Role
  User data

- Notes on Contents:

  The role parameter indicates which of Client, Server, Sender or Receiver is attempting to re-establish the association. The other parameters are the same as for the CON Connect message. However the values of Called Address, Class of Service and Quality of Service must be the same as those negotiated for the prior connection.

- Sending Actions:

  Sent following a valid R-RECONNECT Request.

- Receiving Actions:

  If the association can be recovered an R-RECONNECT Indication is delivered to the Server. Otherwise the request is refused using an RCN-R with a Failure diagnostic.

  It is possible for RCN messages to clash. In this case the message from the Receiver takes priority, but a message from a Client wins against one from a Server.

4.8.4 The RCN-R (Reconnect Response) Message

- Source: Client, Server, Sender or Receiver

- Function: to accept or refuse a reconnection request.

- Contents:
  Diagnostic
  Responding Address
  Association Identifier
  Class of R-Service
  Quality of R-Service
  User Data

- Notes on Contents: These are the same as for CON-R.

- Sending Actions:

  Sent either because the recipient of the RCN message cannot establish an entity to receive the R-RECONNECT Indication, or as a result of an R-RECONNECT Response. If the diagnostic indicates failure the connection is not established, otherwise the association has a new connection and both entities are in a resumption state that depends on the state of the service when it was disrupted.

- Receiving Actions:

  An R-RECONNECT Confirm is delivered and the protocol entity is in a re-connected resumption state.

4.8.5 Failure and Recovery States

This section details the possible states of the protocol machines at the time of failure. In all cases at least one machine must attempt reconnection, and this machine is identified, although in some cases machine may do so. The expected action to effect resynchronisation is also explained.

Machine	Regime	State Code	Notes
CL	Connected	F1	Outside transaction, inc STR-P
CL	Transact	F2	In transaction, inc SEC-P
CL	Secure	F3	Idle state
CL	Secure	F4	Pending Commit or Rollback
CL	Transact	F5	Pending Secure and Commit
CL	Transact	F6	Pending Read Table, RDT-P
CL	Transact	F16	Pending Append Table, APT-P
CL	Transact	F7	After RDT, ETR-P
SV	Connected	F1	Outside Transaction
SV	Transact	F2	In transaction,inc RBP-P
SV	Secure	F3	Idle state
SV	Transact	F4	Pending Secure and Commit RP
SN	Sending	F8	Sending, inc RST and RST-P
SN	Sending	F9	(if SV) data sent, unsecure
SN	Sending	F10	(if CL) data sent, unsecure
SN	Sending	F11	Cancelled by sender, CAN-P
RV	Receiving	F8	Rcving, inc RST, RST-P
RV	Receiving	F9	(if CL) ERD, not yet secure
RV	Receiving	F10	(if SV) ETR, not yet secure
RV	Receiving	F11	Cancelled by receiver, CAN-P

Table 4.8.1 Failure States

The table above lists the failure states. The same table appears in the formal description, Appendix E. The text below lists the combinations of failure states that can occur, and the actions needed to effect resynchronisation.

Combination Number: 1   Brief Description: CL.F1 SV.F1

Explanation: Both machines outside a transaction.

Resynchronisation Sequence:

Client attempts reconnection; there is no guarantee that the last message sent was delivered so the Client must repeat it and may get a failure or warning diagnostic.

Combination Number: 2   Brief Description: CL.F1 SV.F2

Explanation: Client has attempted to start a transaction, and has succeeded, but has not received confirmation.
Resynchronisation Sequence

Client attempts reconnection; the Start Transaction is repeated. The Server aborts its current (and unstarted) transaction and starts the new one.

Combination Number: 3   Brief Description: CL.F2 SV.F2

Explanation: Failure during a transaction, but outside a bulk transfer.

Resynchronisation Sequence

Note that the DBCS may rollback the transaction and release any (implicitly) locked resources as soon as the failure is notified to it. The Client attempts reconnection, and follows this with an immediate Rollback.

Combination Number: 4   Brief Description: CL.F2 SV.F3

Explanation: Client has Secure Pending, Server is Secure.

Resynchronisation Sequence

Client and Server both attempt reconnection. If the Client succeeds, and it would win a clash, it will rollback the transaction. If the server succeeds it will repeat the SEC-R and the Client may either Commit or Rollback the transaction.

Combination Number: 5   Brief Description: CL.F3 SV.F3

Explanation: Failure when a transaction is Secure.

Resynchronisation Sequence

The Server attempts reconnection and repeats the SEC-R.

Combination Number: 6   Brief Description: CL.F4 SV.F3

Explanation: Client has secured the transaction, but now has a Commit or a Rollback pending; the Server is Secure.

Resynchronisation Sequence

Both Client and Server attempt Reconnection, if there is a clash the Client wins. The Client repeats the Commit or Rollback, which is acted upon, or if the server reconnects it repeats the SEC-R and causes the Client to repeat the Commit or Rollback.

Combination Number: 7   Brief Description: CL.F4 SV.F1

Explanation: Client has secured the transaction and is waiting for confirmation of a Commit or Rollback. Server has completed and is outside the transaction context.

Resynchronisation Sequence

Client attempts reconnection and repeats the Commit or Rollback. The server responds with a Commit or Rollback response with a Warning Diagnostic.

Combination Number: 8   Brief Description: CL.F5 SV.F2

Explanation: Client has Secure-and-Commit pending, Server is in normal Transaction context.

Resynchronisation Sequence

Client attempts reconnection and repeats the Secure-and-Commit. This is delivered to the Server.

Combination Number: 9   Brief Description: CL.F5 SV.F1

Explanation: Client has Secure-and-Commit Pending, Server has completed the Transaction.

Resynchronisation Sequence

The Client attempts reconnection and repeats the Secure-and-Commit. The server repeats its previous response confirming the fate of the transaction.

Combination Number: 10   Brief Description: CL.F6 SV.F2

Explanation: Client has Read Table pending, Server is still within transaction and not in bulk transfer mode.

Resynchronisation Sequence

The Client attempts reconnection. It then issues a CAN Cancel which is responded to with a CAN-R. Both machines are then in Transaction Idle state with the Client in control.

Combination Number: 11 Brief Description: CL.F6 SN.F8

Explanation: Client's Read Table has not yet been confirmed, but the Server has already entered the Sending state.

Resynchronisation Sequence

The Client attempts reconnection. It then issues a CAN Cancel which is responded to with a CAN-R. Both machines are then in Transaction Idle state with the Client in control.

Combination Number: 12 Brief Description: CL.F16 SV.F2

Explanation: Client is waiting for response to Append Table, Server has not yet received or acted upon it.

Resynchronisation Sequence

The Client attempts reconnection. It then issues a CAN Cancel which is responded to with a CAN-R. Both machines are then in Transaction Idle state with the Client in control.

Combination Number: 13 Brief Description: CL.F16 RV.F8

Explanation: Client is waiting for response to Append Table, Server has entered Receiving state.

Resynchronisation Sequence

The Client and Receiver both attempt reconnection. If the Client succeeds it then issues a CAN Cancel which is responded to with a CAN-R. Both machines are then in Transaction Idle state with the Client in control. However if the Receiver wins, the Client will be in Sending state and the Receiver will issue a RST Restart.

Combination Number: 14 Brief Description: SN.F8 RV.F8

Explanation: Bulk transfer, and possibly restarting, are in progress at both ends of the connection.

Resynchronisation Sequence

The receiver must attempt reconnection, and the sender may do so if it is also the Client. The Receiver wins if there is a clash.

After reconnection either a Restart is negotiated or the transfer is cancelled.

Combination Number: 15 Brief Description: SN.F8 RV.F11

Explanation: Sender is sending but the Receiver has cancelled.
Resynchronisation Sequence

The receiver must attempt reconnection, and the sender may do so if it is also the Client. The Receiver wins if there is a clash. The Receiver's action depends on its identifying the remote entity as the Sender and therefore knowing that the Cancel message had not been received. In this case it ignores any attempt to restart and issues another CAN.

Combination Number: 16 Brief Description: RV.F11 CL.F2

Explanation: The Receiver has cancelled, but this has not been confirmed, although the CAN has caused the Sender to revert to the Client.

Resynchronisation Sequence

The Receiver attempts reconnection, and so does the Client. In this special case the Client wins if there is a clash, and the Receiver reverts to Transaction Idle status. However if the Receiver succeeds in establishing connection its subsequent CAN will be responded to with a CAN-R; the Client will then Rollback the transaction.

Combination Number: 17 Brief Description: RV.F11 SV.F2

Explanation: The Receiver has cancelled, but this has not been confirmed, although the CAN has caused the Sender to revert to the Server.

Resynchronisation Sequence

The Receiver attempts reconnection and issues a CAN which is responded to with a CAN-R. Both entities are then in Transaction Idle state.

Combination Number: 18 Brief Description: SN.F11 RV.F8

Explanation: Sender has cancelled but the Receiver has not received the CAN and is still receiving.

Resynchronisation Sequence

The receiver must attempt reconnection, and the sender may do so if it is also the Client. The Receiver wins if there is a clash. The receiver will attempt a Restart and be met with a CAN. If the Sender establishes the connection it will immediately reissue the CAN.

Combination Number: 19 Brief Description: SN.F11 CL.F2

Explanation: Sender has cancelled but not received confirmation, Receiver has already reverted to Client, Transaction Idle.

Resynchronisation Sequence

Client and Sender both attempt reconnection, the Client wins if there is a clash. If the Sender establishes the connection it will issue a CAN, which is confirmed by the Client. If the Client establishes the connection the Sender reverts to the Server, Transaction Idle state.

Combination Number: 20 Brief Description: SN.F11 SV.F2

Explanation: The Sender has cancelled but has not received confirmation, and the Receiver has reverted to the Server, Transaction Idle state.

Resynchronisation Sequence

The Sender attempts reconnection and repeats the CAN, which results in a CAN-R and returns the Sender to the Client role.

Combination Number: 21 Brief Description: SN.F9 RV.F8

Explanation: Sender (server) has sent all the data, but the receiver has not received the ERD.

Resynchronisation Sequence

Receiver attempts reconnection. The Sender awaits End Transfer, but the receiver will request Restart.

Combination Number: 22 Brief Description: SN.F10 RV.F8

Explanation: Sender (client) has sent all the data, but the receiver has not received the ETR.

Resynchronisation Sequence

Receiver and sender may both attempt reconnection. If there is a clash the receiver wins. If the receiver connects it requests Restart and should succeed. If the Sender connects it repeats the ETR. This is delivered as an End Transfer Indication, but there may have been data loss and the receiver should request a Restart.

Combination Number: 23   Brief Description: SN.F9 RV.F9

Explanation: Sender (server) has sent End Read, the Receiver has received the Indication and may or may not have secured the data.

Resynchronisation Sequence
The Receiver attempts reconnection and awaits the End Transfer, or possibly Restart, Request from the Service user.

Combination Number: 24   Brief Description: SN.F10 RV.F10

Explanation: the Sender (Client) has sent the ETR End Transfer. The indication has been delivered and the server may or may not have secured the data.

Resynchronisation Sequence

Sender and Receiver may both attempt reconnection. If they clash the Receiver wins. When the Receiver connects it delivers an End Transfer Indication to the Server. This may result in an End Transfer Response or in a Restart Request. If the Sender makes the connection it repeats the ETR.

Combination Number: 25   Brief Description: SN.F10 SV.F2

Explanation: Sender (Client) has sent the ETR End Transfer, but not received confirmation, but the receiver has already reverted to the Server role.

Resynchronisation Sequence

The Sender attempts reconnection and repeats the ETR. The server replies with an ETR-R, Success with Warning.

Combination Number: 26   Brief Description: SN.F9 CL.F7

Explanation: Client has secured a Read Table and sent an ETR, the Sender (Server) is still awaiting the ETR.

Resynchronisation Sequence

The Client reconnects and repeats the ETR.

Combination Number: 27   Brief Description: CL.F7 SV.F2

Explanation: The Client has sent the End Transfer and is waiting for a response. This has been sent but not received.

Resynchronisation Sequence

The Client reconnects and repeats the ETR. This gets the response Success with Warning.

4.9 THE DIAGNOSTIC PARAMETER

The diagnostic parameter consists of two components, the Severity and the Reason Code. The severity indicates relevant classes of error as follows:

0	Indicates Success
1	Indicates Success with Warning
2	Indicates Recoverable Error
3	Indicates Uncorrectable Error

An uncorrectable error is one that will reoccur if the sequence of events that led up to the failure is repeated, and implies at least the failure of the operation being performed.

A recoverable error also implies a failure in the present operation, but it is possible that the error may not reoccur if the sequence is repeated.

A warning does not require recovery and leaves the protocol machine in the same state as success.

The second part of the diagnostic is the Reason Code. Appendix F contains a list of possible values and their meanings.

4.10 PRESENTATION SERVICE MAPPINGS

This section defines the representation of all messages and data types communicated by the Presentation Service on behalf of the RDAP.

The Presentation Service provides several alternative ways of transferring data between its users. For each RDAP message it is necessary to define the presentation service element to be used and the form of data within this service element. It is also necessary to ensure that each message type can be distinguished from messages of other types that may be carried using the same service element.

The content of messages is defined using the Abstract Syntax Notation One (ASN.1), see also Appendix D. The concrete transfer syntax, which determines the possible representations for messages, is determined by applying a set of encoding rules to the ASN.1 message definition. Such a set of encoding rules has been defined by ISO in ISO DP 8825. These encoding rules enable any data type defined in ASN.1 to be encoded and to be unambiguously recognised by the receiver.

Table 4.10.1 below, defines the mapping of RDAP messages to Presentation Service primitives.

RDAP Message	Presentation Service Mapping
CON	P-TYPED-DATA Request
CON-R	P-TYPED-DATA Request
REL	P-RELEASE Request
REL-R	P-RELEASE Response
BGG	P-TYPED-DATA Request - start
MDF	typed data (see below)
MDF-R	typed data
MDR	typed data
MDR-R	typed data
RDL	typed data
RDL-R	typed data
typed data	{ Within group - append to P-TYPED-DATA
	{ Request message
	{ Outside group - P-TYPED-DATA Request
SPR	P-TYPED-DATA Request
EGR	P-TYPED-DATA Request - terminate
EGR-R	P-TYPED-DATA Request - terminate
STR	P-SYNC-MINOR Request, User Data
STR-R	P-SYNC-MINOR Response, User Data
SEC	P-SYNC-MAJOR Request, User Data
SEC-R	P-SYNC-MAJOR Response, User Data
SCM	P-SYNC-MAJOR Request, User Data
SCM-R	P-SYNC-MAJOR Response, User Data
COM	P-TYPED-DATA Request
COM-R	P-TYPED-DATA Request
RBK	P-RESYNCHRONISE Request, User Data
RBK-R	P-RESYNCHRONISE Response, User Data
RBP	P-TYPED-DATA Request
APT	P-TYPED-DATA Request
APT-R	P-TYPED-DATA Request
RDT	P-TYPED-DATA Request
RDT-R	P-TYPED-DATA Request
ERD	P-TYPED-DATA Request
ETR	P-SYNC-MINOR Request, User Data
ETR-R	P-SYNC-MINOR Response, User Data
DAT	typed data
CHK	P-SYNC-MINOR Request, User Data
CHK-R	P-SYNC-MINOR Response, User Data
CAN	P-RESYNCHRONISE Request, User Data
CAN-R	P-RESYNCHRONISE Response, User Data
RST	P-RESYNCHRONISE Request, User Data
RST-R	P-RESYNCHRONISE Response, User Data
DIS	P-U-ABORT
RCN	P-TYPED-DATA Request
RCN-R	P-TYPED-DATA Request

Table 4.10.1 Mapping of RDAP Messages to the Presentation Service

The type of Presentation Service primitive carrying an RDAP message assists in the identification of the RDAP message type, but messages carried by the same Presentation Service Primitive are distinguished from each other by the Abstract Syntax. The Abstract Syntax is therefore presented with messages grouped according to the Presentation Service primitive to which they are mapped.

The syntax for statements is closely related to the syntax for the Database Language SQL and follows the definitions for messages.

4.10.1 RDAP Abstract Syntax for Messages

```
RDAP-MESSAGE DEFINITIONS  ::=

BEGIN

P-Typed-Data     ::= CHOICE { [1] Connect,                   -- CON
                         [2] Connect-Response,               -- CON-R
                         [3] Begin-Group,                    -- BGG
                         [4] Define-Macro,                   -- MDF
                         [5] Define-Macro-Response,          -- MDF-R
                         [6] Drop-Macro,                     -- MDR
                         [7] Drop-Macro-Response,            -- MDR-R
                         [8] DL-SQL-Function,                -- RDL
                         [9] DL-SQL-Response,                -- RDL-R
                         [10] Still-Processing,              -- SPR
                         [11] End-Group,                     -- EGR
                         [12] End-Group-Response,            -- EGR-R
                         [13] Commit,                        -- COM
                         [14] Commit-Response,               -- COM-R
                         [15] Rollback-Please,               -- RBP
                         [16] Append-Table,                  -- APT
                         [17] Append-Table-Response,         -- APT-R
                         [18] Read-Table,                    -- RDT
                         [19] Read-Table-Response,           -- RDT-R
                         [20] End-Read,                      -- ERD
                         [21] Data,                          -- DAT
                         [22] Reconnect,                     -- RCN
                         [23] Reconnect-Response    }        -- RCN-R

P-Release-Request  ::= Release                               -- REL

P-Release-Response ::= Release-Response                      -- REL-R

P-Sync-Minor       ::= CHOICE   {
                         [1] Start-Transaction,              -- STR
                         [2] End-Transfer,                   -- ETR
                         [3] Check                 }         -- CHK
```

```
P-Sync-Minor-Response ::= CHOICE  {
                          [1] Start-Transaction-Response, -- STR-R
                          [2] End-Transfer-Response,      -- ETR-R
                          [3] Check-Response    }         -- CHK-R

P-Sync-Major        ::= CHOICE  {
                          [1] Secure,                     -- SEC
                          [2] Secure-and-Commit    }      -- SCM

P-Sync-Major-Response  ::= CHOICE {
                      [1] Secure-Response,                -- SEC-R
                      [2] Secure-and-Commit-Response }    -- SCM-R

P-Resynchronise    ::= CHOICE {
                      [1] Rollback,                       -- RBK
                      [2] Cancel,                         -- CAN
                      [3] Restart    }                    -- RST

P-Resynchronise-Response   ::=  CHOICE {
                      [1] Rollback-Response,              -- RBK-R
                      [2] Cancel-Response,                -- CAN-R
                      [3] Restart-Response   }            -- RST-R

P-U-Abort                 ::= Disconnect


--            Common Data Types               --

Diagnostic                ::= SEQUENCE   {
                           Severity,
                           reason  INTEGER    }

Severity                  ::=  ENUMERATED  {
                           success (0)
                           warning (1)
                           recoverable-Error(2)
                           uncorrectable-Error(3)

--            Names                             --

Identifier            ::= PrintableString --Max 18 characters

Macro-Name            ::= Identifier

Parameter-Name        ::= Identifier

Table-Name            ::= SEQUENCE {
                          CHOICE {Auth-ID ,NULL },
                          Table-Terminal-Name }

Auth-ID               ::= Identifier

Table-Terminal-Name   ::= Identifier
```

```
--              Association Management              --

Connect-Message          ::=  SEQUENCE   {
                              Called-Address     OCTETSTRING
                              Calling-Address    OCTETSTRING
                              Class-of-R-Service
                              Quality-of-R-Service

                              SET    {
                                [1] Identity-of-Client  PrintableString
                                [2] Current-Account     PrintableString
                                [3] User-Data           OCTETSTRING
                                    }   }

Class-of-R-Service       ::= SET { empty }

Quality-of-R-Service     ::= SET {
                                [1] Ccr
                                [2] Bulk-Transfer
                                [3] Grouping      }

Ccr                      ::= ENUMERATED  {
                             no-ccr (0), minimum-recovery (1)
                             commitment (2),
                             distributed-commitment (3)     }

Bulk-Transfer            ::= ENUMERATED  {
                             none (0), without-checks (1),
                             with-checks (2)     }

Grouping                 ::= BOOLEAN

Connect-Response         ::= SEQUENCE   {
                             Diagnostic,
                             CHOICE   {  SEQUENCE  {
                             Responding-Address      OCTETSTRING,
                             Association-Identifier OCTETSTRING,
                             Class-of-R-Service,
                             Quality-of-R-Service,
                             User-Data                   },
                             empty    }     }

Release-Message          ::=  SEQUENCE   {
                             Suspend    BOOLEAN,
                             User-Data  OCTETSTRING    }

Release-Response         ::=  SEQUENCE   {
                             Diagnostic,
                             User-Data  OCTETSTRING    }
```

```
--    Data Definition and Manipulation Management  --

Define-Macro            ::=  SEQUENCE  {
                            Scope-Temporary  BOOLEAN,
                            Interface,
                            Body        }

Interface               ::=  SEQUENCE  {
                            Macro-Name,
                            Update-Parameters  Parameters,
                            Input-Parameters   Parameters,
                            Output-Parameters   Parameters  }

Parameters              ::=  SEQUENCEOF Parameter-Declaration

Parameter-Declaration ::=  SEQUENCE { Parameter-Name, Data-Type }

Data-Type               ::=  CHOICE  { [1] string-Length INTEGER,
                              [2] exact-Numeric-Scale INTEGER
                              [3] approx-Numeric-Precision INTEGER }

Body                    ::=  SEQUENCEOF  Statement

Statement               ::= CHOICE  {
                                Create-Temporary-Table,
                                Close-Statement,
                                Declare-Cursor,
                                Delete-Statement-Positioned,
                                Delete-Statement-Searched,
                                Fetch-Statement,
                                Insert-Statement,
                                Invoke-Statement,
                                Open-Statement,
                                Select-Statement,
                                Test-Statement,
                                Update-Statement-Positioned,
                                Update-Statement-Searched      }

Define-Macro-Response ::=  SEQUENCE  {
                              COMPONENTSOF  Diagnostic,
                              Extended-Diagnostic       }

Extended-Diagnostic   ::=  SEQUENCEOF {
                              statement-Number  INTEGER,
                              element-Number    INTEGER,
                              fault-Code        INTEGER  }

-- Fault Code Convention +ve  = syntax          --
--                      0    = unknown         --
--                      -ve  = no access       --
```

```
Drop-Macro              ::=  SEQUENCE  {
                               scope-Temporary  BOOLEAN,
                               Macro-Name      }

Drop-Macro-Response     ::=  Diagnostic

DL-SQL-Function         ::=  SEQUENCE  {
                               Statement,
                               Data-Values  }

Data-Values             ::=  SEQUENCEOF ANY    -- values from the
                                               -- database

DL-SQL-Response         ::=  SEQUENCE  {
                               COMPONENTSOF Diagnostic,
                               CHOICE  {
                               [1] Extended-Diagnostic,
                               [2] Data-Values      }    }

Still-Processing        ::=  time INTEGER

--       Transaction Management and Data Manipulation      --

Start-Transaction          ::=  Transaction-ID

Transaction-ID             ::=  OCTETSTRING

Start-Transaction-Response   ::=  empty

Secure                     ::=  empty

Secure-Response            ::=  SEQUENCE  {
                              COMPONENTSOF Diagnostic,
                              Transaction-ID   }

Secure-and-Commit          ::= Transaction-ID

Secure-and-Commit-Response  ::=  Diagnostic

Commit                     ::= empty

Commit-Response            ::= empty

Rollback                   ::= empty

Rollback-Response          ::= Diagnostic

Rollback-Please            ::= Diagnostic
```

```
--                  Bulk   Transfer                      --

Append-Table                 ::= Table-Name

Append-Table-Response        ::= Diagnostic

Read-Table                   ::= CHOICE  { Table-Name,
                                           Select Expression }

Read-Table-Response          ::= Diagnostic

End-Read                     ::= empty

End-Transfer                 ::= CHOICE { Diagnostic,
                                 empty            }

End-Transfer-Response        ::= CHOICE { Diagnostic,
                                 empty            }

Data                         ::= SEQUENCE    {
                                 Rows  INTEGER,
                                 SEQUENCEOF  {Row }   }

Row                          ::= SEQUENCEOF   Column-Value

Column-Value                 ::=  ANY

Check                        ::=  Checkpoint

Checkpoint                   ::=  INTEGER

Check-Response               ::=  Checkpoint

Cancel                       ::=  Diagnostic

Cancel-Response              ::=  Diagnostic

Restart                      ::=  Restart-Point

Restart-Point                ::=  INTEGER

Restart-Response             ::=  SEQUENCE  {
                                  COMPONENTSOF Diagnostic,
                                  Restart-Point   }

 --                  Grouping              --

Begin-Group                  ::= empty

End-Group                    ::= empty

End-Group-Response           ::= empty
```

```
--                   Failure and Recovery of the Service      --

Disconnect                  ::= SEQUENCE  {
                                Suspend   BOOLEAN,
                                User-Data  OCTETSTRING  }

Reconnect                   ::=  Connect

Reconnect-Response          ::=  Connect-Response

END
```

4.10.2 Abstract Syntax for RDAP Statements

```
RDAP-STATEMENTS             ::=

BEGIN

Statement         ::= CHOICE { [1]  Create-Temporary-Table,
                               [2]  Close-Statement,
                               [3]  Declare-Cursor,
                               [4]  Delete-Statement-Positioned,
                               [5]  Delete-Statement-Searched,
                               [6]  Fetch-Statement,
                               [7]  Insert-Statement,
                               [8]  Invoke-Statement,
                               [9]  Open-Statement,
                               [10] Select-Statement,
                               [11] Test-Statement,
                               [12] Update-Statement-Positioned,
                               [13] Update-Statement-Searched      }

Create-Temporary-Table   ::= SEQUENCE  { Table-Name,
                                SEQUENCEOF SEQUENCE
                                    {Column-Name, Data-Type  }  }

Close-Statement             ::=  Cursor-Name

Declare-Cursor              ::=  SEQUENCE { Cursor-Name,
                                            Select-Expression }

Delete-Statement-Positioned  ::= SEQUENCE { Table-Name,
                                    Cursor-Name }

Delete-Statement-Searched    ::= SEQUENCE { Table-Name,
                             CHOICE { Search-Condition, empty } }

Fetch-Statement             ::= SEQUENCE { Cursor-Name,
                                  Parameter-List }

Parameter-List              ::= SEQUENCEOF Parameter-Name
```

```
-- Within an RDL, Parameter-List is empty.  It is used in Macros

Insert-Statement          ::= SEQUENCE {Table-Name,
                         CHOICE { Column-List, NULL }
                         CHOICE { Value-List, Query-Specification }
                          }

Invoke-Statement          ::= SEQUENCE { Macro-Name, SEQUENCEOF
                           parameter-Value  ANY  }

Open-Statement            ::= Cursor-Name

Select-Statement          ::= SEQUENCE  { All BOOLEAN, Select-List,
                              Parameter-List, Table-Expression }

Test-Statement            ::=SEQUENCEOF CHOICE {
                           [1] error-code INTEGER,
                           [2] error-range SEQUENCE {
                           low INTEGER, high INTEGER  }  }

Update-Statement-Positioned  ::= SEQUENCE  {
                       Table-name, Cursor-name,
                       SEQUENCEOF Set-Clause      }

Update-Statement-Searched    ::= SEQUENCE  {  Table-Name,
                       SEQUENCEOF Set-Clause,
                       CHOICE    { Search-Condition, NULL }  }


--                 Common Data Types          --


--                Names                 --

Identifier                ::= Printable String              -- Max 18

Cursor-Name               ::= Identifier

Macro-Name                ::= Identifier

Parameter-Name            ::= Identifier

Table-Name                ::= SEQUENCE {
                           CHOICE  {Auth-ID ,NULL },
                           Table-Terminal-Name  }

Auth-ID                   ::= Identifier

Table-Terminal-Name       ::= Identifier

Correlation-Name          ::= Identifier

Column-Name               ::= Identifier
```

```
--                Data Type                       --

   Data-Type                ::= CHOICE  { [1] string-Length INTEGER,
                                  [2] exact-Numeric-Scale INTEGER,
                                  [3] approx-Numeric-Precision INTEGER }

   --               Selection                       --

   Select-Expression        ::= SEQUENCE   {
                                SEQUENCEOF Query-Term,
                                SEQUENCEOF Sort-Spec         -- may be --
                                           }                 --  empty --

   Query-Term               ::= CHOICE {
                                Query-Spec,
                                SEQUENCEOF Query-Term   }

   Sort-Spec                ::= SEQUENCE { asc BOOLEAN ,
                                CHOICE { INTEGER, Column-Spec     }      }

   Query-Spec               ::= SEQUENCE {
                                distinct BOOLEAN,
                                Select-List,
                                Table-Expression }

   Select-List              ::= CHOICE {
                                SEQUENCE OF Value-Expression,
                                all NULL   }

   Table-Expression         ::= SEQUENCE {
                                SEQUENCEOF Table-Reference,
                                SET { [1] Search-Condition  OPTIONAL,
                                          --    Where Clause   --
                                      [2] SEQUENCEOF Column-Spec
                                          OPTIONAL,   -- Group by  --
                                      [3] Search-Condition  OPTIONAL }
                                          --    Having Clause   --      }

   Table-Reference          ::= SEQUENCE   {
                                Table-Name,
                                CHOICE { Correlation-Name, NULL}    }

   Column-Spec              ::= SEQUENCE { CHOICE   { Correlation-Name,
                                                NULL },
                                           Column-Name }

   Search-Condition         ::= CHOICE { Boolean-Term, [3] SEQUENCEOF
                                                  Boolean-Term }

   Boolean-Term             ::= CHOICE { Boolean-Factor, [2] SEQUENCEOF
                                                  Boolean-Factor }
```

```
Boolean-Factor          ::= SEQUENCE { not BOOLEAN, Boolean-Primary}

Boolean-Primary         ::= CHOICE { [1] Search-Condition,
                            COMPONENTSOF Predicate }

Predicate               ::= CHOICE {
                            [2] Comparison,
                            [3] Between,
                            [4] Quantified,
                            [5] Isin,
                            [6] Like,
                            [7] Exists,
                            [8] Null      }

Comparison              ::= SEQUENCE { Comp-op, Value-Expression,
                            CHOICE { Value-Expression, Sub-Query } }

Comp-op                 ::= ENUMERATED { eq(1), lt(2), le(3), gt(4),
                                         ge(5), ne(6) }

Between                 ::= SEQUENCE  { not    BOOLEAN,
                             subject       Value-Expression,
                             low-value     Value-Expression,
                             high-value    Value-Expression }

Quantified              ::= SEQUENCE { all   BOOLEAN,
                            Comp-op,
                            Value-Expression,
                            Multi-Set-Value-Expression  }

Isin                    ::= SEQUENCE { not   BOOLEAN,
                            Value-Expression,
                            Multi-Set-Value-Expression  }

Like                    ::= SEQUENCE { not   BOOLEAN,
                            Column-Spec,  OCTETSTRING  }

Exists                  ::= Sub-Query

Null                    ::= SEQUENCE { not BOOLEAN, Column-Spec }

Sub-Query               ::= SEQUENCE {
                            distinct BOOLEAN,
                            CHOICE  { [1] Value-Expression, [2] NULL
                                         -- i.e. all --  }
                            Table-Expression }

Value-Expression        ::= CHOICE { Term, [1] SEQUENCE
                               { Term, SEQUENCEOF SEQUENCE
                                   { minus BOOLEAN, Term }  }  }
```

```
Term                   ::= CHOICE { Factor [2] SEQUENCE
                               { Factor, SEQUENCEOF SEQUENCE
                                   { recip BOOLEAN, Factor }  }  }

Factor                 ::= SEQUENCE { minus BOOLEAN, Primary }


Primary                ::= CHOICE { COMPONENTSOF Value-Spec,
                                [5] Column-Spec,
                                COMPONENTSOF Function-Spec,
                                [9] Value-Expression }

Value-Spec             ::= CHOICE { [1] Parameter-Name,
                                [2] Embedded-Variable-Name,
                                [3] literal  ANY,
                                [4] user  BOOLEAN, NULL }

Column-Spec            ::= SEQUENCE { CHOICE
                           { Table-Name, NULL }, Column-Name }
                           -- correlation --

Function-Spec          ::= CHOICE { [6] count NULL,
                           [7] All-Function,
                           [8] Distinct-Function }

All-Function           ::= SEQUENCE { Function, Value-Expression }

Distinct-Function      ::= SEQUENCE { Function, Column-Spec }

Function               ::= ENUMERATED  {  average (1), max (2),
                           min (3), sum (4), count (5) }
```

APPENDICES

APPENDIX A

BRIEF DESCRIPTION OF THE REFERENCE MODEL OF OPEN SYSTEMS INTERCONNECTION

A.1 SCOPE

This appendix is not an integral part of the standard.

This appendix provides a brief description of the Reference Model of Open Systems Interconnection.

A.2 GENERAL DESCRIPTION

A2.1 Introduction

The Reference Model of Open Systems Interconnection provides a common basis for the co-ordination of the development of new standards for the interconnection of systems and also allows existing standards to be placed within a common framework. The model is concerned with systems comprising terminals, computers and associated devices and the means of transferring information between these systems.

A.2.2 Overall perspective

The model does not imply any particular systems implementation, technology or means of interconnection, but rather refers to the mutual recognition and support of the standardized information exchange procedures.

A.2.3 The Open Systems Interconnection environment

Open Systems Interconnection is not only concerned with the transfer of information between systems (i.e. with communication), but also with the capability of these systems to interwork to achieve a common (distributed) task. The objective of Open Systems Interconnection is to define a set of standards which allow interconnected systems to co-operate.

The Reference Model of Open Systems Interconnection recognizes three basic constituents (see fig. 1):

- application processes within an OSI environment,
- connections which permit information exchange,
- the systems themselves.

Note A.1

The application processes may be manual, computer or physical processes.

A.2.4 Management Aspects

Within the Open Systems Interconnection architecture there is a need to recognize the special problems of initiating, terminating, and monitoring on-going activities and assisting in their harmonious operations as well as handling abnormal conditions. These have been collectively considered as the management aspects of the Open Systems Interconnection architecture. These concepts are essential to the operation of the interconnected open systems and therefore are included in the comprehensive description of the Reference Model.

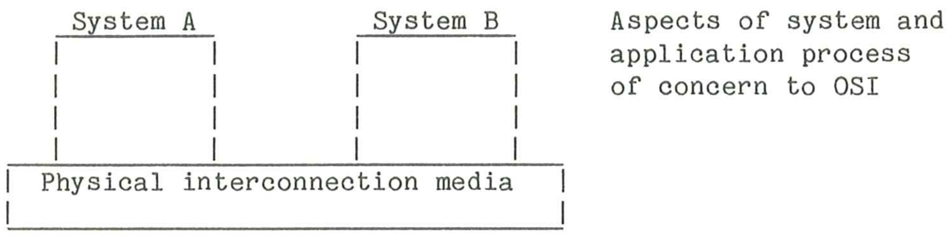
System A
System B
Aspects of system and application process of concern to OSI
Physical interconnection media

Fig. 1 - General schematic diagram illustrating the basic elements of Open Systems Interconnection.

A.2.5 Concepts of a Layered Architecture

The open systems architecture is structured in Layers. Each system is composed of an ordered set of sub-systems represented for convenience by Layers in a vertical sequence. Adjacent subsystems communicate through their common interface.

A Layer consists of all subsystems with the same rank. The operation of a layer is the sum of the co-operation between entities in that Layer. It is governed by a set of protocols specific to that Layer.

The services of a Layer are provided to the next higher Layer, using the functions performed within the Layer and the services available from the next lower Layer.

An entity in a Layer may provide services to one or more entities in the next high Layer and use the services of one or more entities in the next lower Layer.

A.3 THE LAYERED MODEL

The seven-Layer Reference Model is illustrated in fig.2.

```
LAYER                                peer-to-peer protocol
                                              |
                                             \/
                         ________                    ________
Application      -->    |        | <-----------------> |        |
                        |________|                    |________|
Presentation     -->    |        | <-----------------> |        |
                        |________|                    |________|
Session          -->    |        | <-----------------> |        |
                        |________|                    |________|
Transport        -->    |        | <-----------------> |        |
                        |________|                    |________|
Network          -->    |        | <-----------------> |        |
                        |________|                    |________|
Link             -->    |        | <-----------------> |        |
                        |________|                    |________|
Physical         -->    |        | <-----------------> |        |
                        |        |                    |        |
                   ______________________________________________
                          /\
                   ______________________________________________
                          |
                           Physical media for interconnection
```

Fig. 2 - The seven-layer Reference Model and peer-to-peer protocol.

A.3.1 The Application Layer

As the highest layer in the Reference Model of Open Systems Interconnection, the Application Layer provides services to the users of the OSI environment, not to a next higher layer.

The purpose of the Application Layer is to serve as the window between communicating users of the OSI environment through which all exchange of meaningful (to the users) information occurs.

The user is represented by the application-entity to its peer.

All user specifiable parameters of each communications instance are made known to the OSI environment (and thus to the mechanisms implementing the OSI environment) via the Application Layer.

A.3.2 The Presentation Layer

The purpose of the Presentation Layer is to represent information to communicating application-entities in a way that preserves meaning while resolving syntax differences.

The nature of the boundary between the Application Layer and the Presentation Layer is different from the nature of other Layer boundaries in the architecture.

The following principles are adopted to define a boundary between the Application Layer and the Presentation Layer:

- internal attributes of the virtual resource and its manipulation functions exist in the Presentation Layer;
- external attributes of the virtual resource and its manipulation functions exist in the Application Layer;
- the functions to use the services of the Session Layer effectively exist in the Presentation Layer;
- the functions to use services of the Presentation Layer effectively exist in the Application Layer.

A.3.3 The Session Layer

The purpose of the Session Layer is to provide the means necessary for cooperating presentation-entities to organize and synchronize their dialogue and manage their data exchange. To do this, the Session Layer provides services to establish a session-connection between two presentation entities, and to support their orderly data exchange interactions.

To implement the transfer of data between the presentation-entities, the session-connection is mapped onto and uses a transport-connection.

A.3.4 The Transport Layer

The Transport Layer exists to provide the transport-service in association with the underlying services provided by the supporting layers.

The transport-service provides transparent transfer of data between session entities. Transport Layer relieves the transport users from any concern with the detailed way in which reliable and cost effective transfer of data is achieved.

The Transport Layer is required to optimize the use of the available communication resources to provide the performance required by each communicating transport user at minimum cost. This optimization will be achieved within the constraints imposed by considering the global demands of all concurrent transport users and the overall limit of resources available to the Transport Layer. Since the network service provides network connections from any transport entity to any other, all protocols defined in the Transport Layer will have end-to-end significance, where the ends are defined as the correspondent transport-entities.

The transport functions invoked in the Transport Layer to provide requested service quality will depend on the quality of the network service. The quality of the network service will depend on the way the network service is achieved.

A.3.5 The Network Layer

The Network Layer provides the means to establish maintain and terminate network connections between systems containing communicating application-entities and the functional and procedural means to exchange network service data units between two transport entities over network connections.

A.3.6 The Link Layer

The purpose of the Link Layer is to provide the functional and procedural means to activate, maintain and deactivate one or more data link connections among network entities.

The objective of this layer is to detect and possibly correct errors which may occur in the Physical Layer. In addition, the Link Layer conveys to the Network Layer the capability to request assembly of data circuits within the Physical Layer (i.e. the capability of performing control of circuit switching).

A.3.7 The Physical Layer

The Physical Layer provides mechanical, electrical, functional and procedural characteristics to activate, maintain and deactivate physical connections for bit transmission between data link entities possibly through intermediate systems, each relaying bit transmission within the Physical Layer.

APPENDIX B

DATABASE LANGUAGE SQL - SYNTAX SUMMARY

B.1 INTRODUCTION

This appendix contains a summary of the DL SQL syntax supported by the RDA specification, in BNF notation. This syntax is an extension of the syntax in the current ISO DL SQL specification.

B.2 THE DL SQL SYNTAX

```
<macro spec>          ::= MACRO <scope> <interface spec> <body spec>

<scope>               ::= TEMPORARY | PERMANENT

<interface spec>      ::= <macro name>
                          [ UPDATE <parameter spec>...]
                          [ IN   <parameter spec>...   ]
                          [ OUT <parameter spec>...    ]

<parameter spec>      ::= <parameter name> <data type>

<body spec>           ::= <statement>... END

<statement>           ::= <create temporary table> |
                          <close statement> |
                          <commit statement> |
                          <cursor declaration> |
                          <delete statement: positioned> |
                          <delete statement: searched> |
                          <fetch statement> |
                          <insert statement> |
                          <invoke statement > |
                          <open statement> |
                          <rollback statement> |
                          <secure statement> |
                          <secure and commit statement> |
                          <select statement> |
                          <test statement> |
                          <update statement: positioned> |
                          <update statement: searched>

<create temporary table> ::= CREATE TEMPORARY TABLE <table name>
                                 <column definition>...

<column definition>   ::= <column name> <data type>

<close statement>     ::= CLOSE <cursor name>
```

```
<commit statement>    ::= COMMIT WORK

<cursor declaration> ::= DECLARE <cursor name> CURSOR FOR <select
                         expression>

<delete statement: positioned>   ::= DELETE FROM <table name> WHERE
                         CURRENT OF <cursor name>

<delete statement: searched> ::= DELETE FROM <table name> [WHERE
                         <search condition>]

<fetch statement>     ::= FETCH <cursor name> INTO <parameter list>

<insert statement>    ::= INSERT INTO <table name> [(<column list>)]
                         {VALUES <value list> | <query specification>}

<invoke statement>    ::= INVOKE <macro name> USING <value list>

<open statement>      ::= OPEN <cursor name>

<rollback statement> ::= ROLLBACK WORK

<select statement>    ::= SELECT [ ALL | DISTINCT ] <select list>
                         INTO <parameter list> <table expression>

<secure statement>    ::= SECURE

<secure and commit statement> ::= SECURE AND COMMIT

<test statement>      ::= IF <error condition> THEN <action>

<error condition>     ::= STATUS <range>

<range>               ::= <element> | <element> , <range>

<element>             ::= <number> |  <number> THRU <number>

<action>              ::= CONTINUE |  RETURN

<update statement: positioned> ::= UPDATE <table name> SET <set clause>
                         [{,<set clause>}...] WHERE CURRENT OF <cursor
                         name>

<update statement: searched>   ::= UPDATE <table name> SET <set clause>
                         [{,<set clause>}...] [WHERE <search condition>]

<set clause>          ::= <column name> = {<value expression> | NULL }

<select expression>  ::= <query expression> [<order clause>]

<query expression>   ::= <query term> | <query expression> UNION
                                       [ALL] <query term>
```

```
<query term>          ::= <query specification> | (<query expression>)

<order clause>        ::= ORDER BY <sort specification> [{,<sort
                          specification> }...]

<sort specification> ::= {<unsigned integer>  |<column specification>}
                          [ ASC | DESC ]

<query specification> ::= SELECT [ALL | DISTINCT] <select list> <table
                           expression>

<select list>         ::=  <value expression> [,<value expression> ]... | *

<table expression>    ::=  <from clause> [<where clause>] [<group by
                            clause>] [<having clause>]
<from clause>         ::= FROM <table reference> [{,<table reference>}]

<table reference>     ::= <table name> [<correlation name>]

<where clause>        ::= WHERE <search condition>

<group by clause>     ::= GROUP BY <column specification>
                          [,<column specification>]...

<having clause>       ::= HAVING <search condition>

<search condition>    ::= <boolean term> | <search condition> OR  <boolean
                           term>

<boolean term>        ::= <boolean factor> | <boolean term> AND <boolean
                              factor>

<boolean factor>      ::= [NOT] <boolean primary>

<boolean primary>     ::= <predicate> | (<search condition>)

<predicate>           ::= <comparison predicate> | <between predicate> |
                          <quantified predicate> | <in predicate> |
                          <like predicate> | <exists predicate> |
                          <null predicate>

<comparison predicate> ::= <value expression> <comp op> {<value
                            expression> | <sub-query>}

<comp op>             ::= = | <> | < | > | <= | >=

<between predicate>   ::= <value expression> [NOT] BETWEEN <value
                             expression> AND <value expression>

<quantified predicate> ::= <value expression> <comp op>
                             [ALL | ANY | SOME ]  <sub-query>
```

```
<in predicate>         ::= <value expression> [NOT] IN {<sub-query> |
                                         <value list>}

<like predicate>       ::= <column specification> [NOT] LIKE
                                         <value specification>

<exists predicate>     ::= EXISTS <sub-query>

<null predicate>       ::= <column specification> IS [NOT] NULL

<sub-query>            ::= (SELECT [ALL |  DISTINCT] <result specification>
                           <table expression>  )

<result specification> ::= <value expression> |  *

<value expression>     ::= <term> | <value expression> {+ | -} <term>

<term>                 ::= <factor> | <term> {* |  /} <factor>

<factor>               ::= [+ | -] <primary>

<primary>              ::= <value specification> | <column specification> |
                           <function specification> | (<value expression>)

<column specification> ::= [<correlation name>.] <column name>

<function specification> ::= COUNT (*) |  <distinct function> |
                               <all function>

<distinct function>    ::= { AVG | MAX | MIN  | SUM | COUNT} (DISTINCT
                                             <column specification> )

<all function>         ::= { AVG | MAX | MIN | SUM} ([ALL] <value expression>)

<value list>           ::= (<value specification> [,{<value
                                                  specification>}...])

<parameter list>       ::= (<parameter specification> [,{<parameter
                                                  specification>}...])

<value specification> ::= <parameter specification> | <variable
                            specification> | <literal> | USER

<parameter specification> ::= <parameter name> [<indicator parameter>]

<indicator parameter>     ::= <parameter name>

<variable specification> ::= <embedded variable name> [<indicator
                                    variable>]

<indicator variable>      ::= <embedded variable name>
```

```
<data type>            ::= <character string type> | <exact numeric type>
                          | <approximate numeric type>

<character string type>  ::= CHARACTERS <integer>

<exact numeric type>     ::= FIXED <precision>, <scale>

<approximate numeric type> ::= FLOAT <precision>

<precision>              ::= <integer>

<scale>                  ::= <integer>
```

APPENDIX C

NOTATION

C.1 INTRODUCTION AND SCOPE

This Appendix is an integral part of the specification.

This notation is consistent with other ECMA standards and specifications for Open Systems Interconnection.

C.2 DEFINITIONS

This terminology is for the notation defined in this Appendix.

(X) - service: a service element of the remote database access service, of which (X) is its particular name.

(Service) primitive: a discrete component of an (X) - service.

(X) - Request primitive: a type of primitive by means of which a RDA user causes an occurrence of the (X) - service.

(X) - Indication primitive: a type of primitive by means of which a RDA user if informed of an occurrence of the (X) - service.

(X) - Response primitive: a type of primitive by means of which a RDA user replies to an occurrence of an (X) - indication primitive.

(X) - Confirmation primitive: a type of primitive by means of which a RDA user is informed of an occurrence of an (X) - response primitive.

Service structure: the series of one or more primitives of which an (X) - service wholly consists.

Service structure type 1 RI: a service structure with a request primitive and an indication primitive.

Service structure type 2 RC: a service structure with a request primitive, an indication primitive, a response primitive and a confirmation primitive.

Service structure type 3 II: a service structure with two indication primitives.

Event: the occurrence of a primitive.

Initiator: the RDA user who issues the request primitive to the (X) - service concerned.

Acceptor: the RDA user who receives the indication primitive of the (X) - service concerned.

C.3 SERVICE MODEL

The remote database access service is modelled as an abstract service to which RDA users gain access at RDA-access-points. All interactions are between two RDA users located at separate RDA-access-points. A single RDA connection is modelled.

C.4 PRIMITIVES

The remote database access service is defined by means of service primitives.

Primitives are conceptual and are not intended to be directly related to remote database access protocol elements or to the units of interaction across a procedural interface in an implementation. The descriptive technique is independent of such variable details.

Primitives which relate only to local conventions between a RDA user and an implementation are not defined.

The subdivision of the remote database access service into the particular primitives chosen is arbitrary, in that the same remote database access service could be described by other logically equivalent primitives. There is no notion that a primitive is "elementary".

A primitive occurs at one service access point (not both). It usually has parameters, containing values related to its occurrence.

The occurrence of a primitive is a logically instantaneous and indivisible event. The primitive occurs at a logically separate instant, which cannot be interrupted by the occurrence of another primitive. It occurs either completely or not at all.

There are four types of primitives in this standard (see C.2):

a) request primitive
b) indication primitive
c) response primitive
d) confirmation primitive.

The primitives are given names prefixed by "R" to distinguish them from primitives of adjacent layers. The names of the primitives are written in upper case, e.g. R-DATA.

C.5 SERVICE STRUCTURE

Each service element consists of one or more primitives and affects both service access points. There are three different combinations of primitives. These combinations are referred to as service structures. The three service structure types used in this standard are defined in C.2.

Unlike the occurrence of its constituent primitives, the occurrence of a service element is not logically instantaneous and indivisible. The intervals between its constituent primitives may be non-disruptively interspersed with the primitives of other service elements, subject to restrictions particular to the service concerned. Service elements may also be disrupted by the occurrence of certain other primitives (see C.6).

C.6 EFFECTS OF SERVICES

The effects of a service are referred to as being sequentially transmitted if its successive primitives at one service access point result in the same sequence of corresponding primitives at the other service access point (unless disrupted, see below).

The effects of a service are referred to as being disruptive if it may destroy, and therefore prevent the occurrence of, indication and confirmation primitives corresponding to previous request or response primitives. The effects of disrupted services are expedited, unless stated otherwise.

The effects of a primitive are referred to as being non-disruptive if they do not have the above disruptive effects. Non-disruptive effects may include effects relating to or delaying other events without destroying them.

Unless otherwise specified, the effects of a service are sequentially transmitted and non-disruptive.

C.7 PARAMETER NOTATION

For each service element, the parameters are defined by a table followed by a list of parameter descriptions. The column headings in the parameter tables indicate the primitive types: Request, Indic. for Indication, Response, Confirm for Confirmation.

The values in the columns of the parameter tables obey the following conventions:

D (down)	: value supplied by the RDA user in the primitive
U (up)	: value supplied by the RDA service in the primitive

B (both)   : value supplied either by the user or by the RDA service in the primitive

X or blank : parameter not used in the primitive.

The detailed description of a parameter includes its purpose, the rule for setting its value, the default value and the legal values.

Unless otherwise stated, the parameter value in the indication is the same as that in the request, and the parameter value in the confirmation is the same as that in the response.

Parameter values are only defined to a level which distinguishes meaning, but generally without defining their absolute value or encoding. These details are outside the scope of the standard, being local conventions between the RDA user and the implementation.

APPENDIX D

THE ABSTRACT TRANSFER SYNTAX NOTATION

D.1 INTRODUCTION

The notation used in this specification for the definition of the permissible contents of RDAP messages is defined in ISO/DP 8824, Specification of Abstract Syntax Notation One (ASN.1). ASN.1 is a data definition language which permits the specification of data types. A data type is a specification of a class of data values, such as integers and Boolean values. ASN.1 includes the definition of a number of generally useful types and it permits more complex data types to be defined as structures or constructions from other types.

It is important that the receiver of any message should be able to distinguish the data types of the elements that are contained within it. In ASN.1 each data type is assigned a tag, which is an integer value that serves to identify it. The tags are not unique, but they will be unique within the context within which they may occur.

Four classes of tag are specified in the notation.

The first is the UNIVERSAL class. These tags are only defined in the ASN.1 Standard.

The second is the APPLICATION class. These may be defined within the specifications for other standards, such as RDAP.

The third is the PRIVATE class. These tags are enterprise specific.

The final class is the context-specific class. These tags are freely assigned within any use of the notation. They are interpreted according to the context in which they occur.

The data types defined in the ASN.1 specification are shown in table D.1.

Type	Tag
Boolean	UNIVERSAL 1
Integer	UNIVERSAL 2
Bit String	UNIVERSAL 3
Octet String	UNIVERSAL 4
Null	UNIVERSAL 5
Sequence and Sequence of	UNIVERSAL 16
Set and Set of	UNIVERSAL 17
Character Strings	UNIVERSAL 18-22

Table D.1 ASN.1 Data Types

D.2 ASN.1 MACROS FOR RDAP

ASN.1 permits the notation to be extended by the use of Macros. In RDAP the following macros are defined to permit the easy specification of numeric data equivalent to <exact numeric> and <approximate numeric> and to improve the readability of the message definitions (by use of ENUMERATED).

D.2.1 Exact Numeric Data

```
FIXED MACRO ::= BEGIN

TYPE NOTATION ::= "FIXED" "SCALE" sign "number"
                      <Type ::= APPLICATION [1] Integer>

VALUE NOTATION ::= value (VALUE Type) scale

sign           ::= "+" | "-"

scale          ::= "*10**" "number"   | "/10**" "number"

END
```

This macro allows fixed point numbers, i.e. exact numerics, to be declared by specifying the assumed position of the decimal point within the integer value that is the transfer encoding of the value. The value of the scale specifies the number of places to the right of the decimal point. Hence the true value can be computed by dividing the integer representation by ten to the power of the scale.

D.2.2 Approximate Numeric Data

```
FLOAT MACRO ::= BEGIN

TYPE NOTATION ::= "FLOAT" precision base
                    <Type ::= APPLICATION [2] Sequence {Integer,
                              Integer, Integer } >

VALUE NOTATION ::= value (VALUE Type)

precision      ::= "number"

base           ::= "number"

END
```

This macro allows floating point numbers, i.e. approximate numerics, to be declared. They are encoded as a sequence of three integers. The value can be computed by the recipient using the formula,

```
value = a * b **c,
```

where a,b,c are the first, second and third integers in the sequence respectively.

D.2.3 Enumerated Data

```
ENUMERATED MACRO ::= BEGIN

TYPE NOTATION ::= "ENUMERATED" "{" NamedNumberList "}"
                     <Type ::= APPLICATION [3] Integer>

VALUE NOTATION ::= value (VALUE Type)

END
```

This macro provides a more readable alternative to the ASN.1 named numbers and makes a clear distinction from numeric integer values.

APPENDIX E

FORMAL DESCRIPTION

E.1 INTRODUCTION

This appendix is an integral part of the specification.

In section 4 of the specification, the RDA protocol interactions between two RDA protocol entities are described. That description references states, events and actions which in this appendix are consolidated into a formal description of the protocol, as a finite state machine.

The formal description identifies the protocol entities and the states that they may assume. It identifies all the events that may occur in each state in a valid implementation and defines the action of the protocol entities in each case.

E.2 PROTOCOL MACHINES

The protocol machines service the requests of the client and server protocol user entities. We distinguish four machine roles:

CL	Client
SV	Server
SN	Sender
RV	Receiver

Both client and server protocol entities may assume the sender or receiver role.

E.3 MACHINE STATES AND REGIMES

For convenience in design and modelling the states of the protocol machines are grouped into regimes. Most service events and messages are only valid in one regime. Table E/1 lists the regimes for each protocol machine.

The action of a protocol machine when it is notified of an event depends on its state at the time. There are four types of machine state:

- idle states	- in which there are no uncompleted service elements;
- pending states	- in which the machine is within a service element, expecting a particular event;
- grouping states	- in which the normal flow is being modified by grouping;
- fail states	- after a disruption in service and during recovery.

E.3.1 Idle States

Table E/1 lists the machines, and the idle states associated with each regime.

Machine	Regime	State Code	State Name
CL	Connected	CNECTED	Connected
CL	Disconnect	IDLE	Idle
CL	Transact	TR-IDLE	Transaction Idle
CL	Secure	SECURE	Transaction Secured
RV	Bulk Tran	RCVING	Bulk Transfer Read
SN	Bulk Tran	SENDING	Bulk Transfer Write
SV	Connected	CNECTED	Connected
SV	Disconnect	IDLE	Idle
SV	Transact	TR-IDLE	Transaction Idle
SV	Transact	SECURE	Transaction Secured

Table E/1 Protocol Machines, Regimes and Idle States

E.3.2 Pending States

Pending states are associated with particular service events, and are related to message types.

The postfixes -P, -PP, -PQ indicate pending states:

XXX-P indicates that the machine is waiting for a reply to message XXX

XXX-PP indicates waiting for a Service Response after XXX indication

XXX-PQ indicates waiting for a Service Request after XXX indication.

The latter state occurs very occasionally, when the service user has no choice over the next service.

E.3.3 Grouping States

The state event tables model the actions when grouping is not being used. The effect of grouping are defined as a modification of the normal behaviour in section E.7 .

E.3.4 Failure States

The normal flow of messages may be interrupted by failure in either of the user nodes or in the Presentation Service. When failure occurs the protocol machine is left in a failure state pending reconnection. There are different failure states, depending on the state of the machines when failure is detected. There are also special states associated with the recovery, prior to resumption of the normal flow.

Many failures leave the service in an uncertain condition. The last message sent may or may not have been received, and if it was received it may or may not have been acted upon.

Table E/2 lists the failure states after disconnection for each protocol machine role.

Machine	Regime	State Code	Notes
CL	Connected	F1	Outside transaction, inc STR-P
CL	Transact	F2	In transaction, inc SEC-P
CL	Secure	F3	Idle state
CL	Secure	F4	Pending Commit or Rollback
CL	Transact	F5	Pending Secure and Commit
CL	Transact	F6	Pending Read Table, RDT-P
CL	Transact	F16	Pending Append Table, APT-P
CL	Transact	F7	After RDT, ETR-P
SV	Connected	F1	Outside Transaction
SV	Transact	F2	In transaction,inc RBP-P
SV	Secure	F3	Idle state
SV	Transact	F4	Pending Secure and Commit RP
SN	Sending	F8	Sending, inc RST and RST-P
SN	Sending	F9	(if SV) data sent, unsecure
SN	Sending	F10	(if CL) data sent, unsecure
SN	Sending	F11	Cancelled by sender, CAN-P
RV	Bulk Tran	F8	Rcving, inc RST, RST-P
RV	Bulk Tran	F9	(if CL) ERD, not yet secure
RV	Bulk Tran	F10	(if SV) ETR, not yet secure
RV	Bulk Tran	F11	Cancelled by receiver, CAN-P

Table E/2 Failure states following disconnection

During reconnection and recovery the reconnect pending states and a special RESUME or RES state generally exist. These states are prefixed by the failure state number, eg F10RCN-P would be the state pending a Reconnect Response message following failure F10.

E.4 EVENTS AFFECTING THE PROTOCOL MACHINES

There are Protocol Machine events for each Service Primitive and for the receipt of each message defined by the protocol. Table E/3 shows the convention for encoding events. These codes are used in the State Event tables.

Event Code	Event Description
XXX	XXX request message
XXX-R	XXX response message
XXXRQ	Request Primitive causing XXX
XXXIN	Indication Primitive for XXX
XXXRP	Response Primitive for XXX-R
XXXCF	Confirmation Primitive for XXX
P-ABORT	Abrupt termination of the P-Service

Table E/3 RDA Protocol Machine Events

E.5 PROTOCOL MESSAGES

Table E/4 lists the message codes for the protocol messages. These codes are used in the State Event tables with the postfixes defined in table E/3 above.

Message Code	Sending Machine	Sending Regime	Reply Message	Description
CON	CL	Disconnect	CON-R	Connect
RCN	CL, SV, SN, RV	Disconnect	RCN-R	Reconnect
REL	CL	Connected	REL-R	Release
DIS	CL, SV	Any		Disconnect
PAB	P-Service	Any		Abort
MDF	CL	Connected	MDF-R	Macro Define
MDR	CL	Connected	MDR-R	Macro Drop
STR	CL	Connected	STR-R	Start Transaction
SEC	CL	Transact	SEC-R	Secure
SCM	CL	Transact	SCM-R	Secure and Commit
RBK	CL	Transact	RBK-R	Rollback
		Secure		
COM	CL	Secure	COM-R	Commit
RBP	SV	Transact		Rollback Please
BGG	CL	See E/6		Begin Group
EGR	CL	See E/6		End Group
RDL	CL	Transact	RDL-R	DL SQL Functions
SPR	SV	Transact		Still Processing
APT	CL	Transact	APT-R	Append Table
RDT	CL	Transact	RDT-R	Read Table
ETR	CL (SN or RV)	Bulk Tran	ETR-R	End Transfer
ERD	SN	Bulk Tran		End Read
CHK	SN	Bulk Tran	CHK-R	Checkpoint
CAN	SN,RV	Bulk Tran	CAN-R	Cancel Transfer
RST	SN,RV	Bulk Tran	RST-R	Restart
DAT	SN	Bulk Tran		Data

Table E/4 Protocol Messages

E.6 STATE EVENT TABLES

The full State Event table has been segmented into a number of smaller tables for ease of presentation.

Each smaller table applies to one of the Client, Server, Sender or Receiver machine roles and generally applies to a single regime.

The component tables are:

The columns represent the states and the rows represent the valid events that may occur when in that regime. Where the intersection of the row and column contains a blank, or where the combination of state and event is not represented, an occurrence of the event in that state represents a protocol error. Protocol errors cause a disconnection of the service.

If level 0 or level 1 CCR Quality of R-Service has been negotiated, the CNECTED state is the same as the TR-IDLE state and none of the transaction management messages (STR, SEC, SCM, COM, RBK, RBP) are valid. Also, the tables defining Fail States do not apply.

The following conventions are used in the tables:

P-Service	- establish a Presentation Service connection if one does not already exist
Message Code	- action, send the message
Service Event	- action, cause the event

+ or +ve	- prior to action - execute the action if the diagnostic is success or warning
- or -ve	- prior to action - execute the action if the diagnostic indicates failure
+ or -	- after message name - send success or failure diagnostic respectively
:	- prior to state name - move to new state
purge	- destroy all queued messages
resync	- request P-service to clear itself
if	- test condition true
&	- logical AND of two conditions
ckct	- count of outstanding checkpoints
ckno	- checkpoint number
CL,SV,SN,RV	- following "if" tests machine role followed by ".", qualifies a state name
R.	- following "if" tests role of remote machine, e.g. "if R.CL" tests whether the remote machine is the Client.

STATE EVENT	IDLE	CON-P				
CONRQ	P-Service CON :CON-P					
CON-R		CONCF +:CNECTED -ve :IDLE				
P-ABORT		CONCF -ve :IDLE				
DISRQ		purge DIS :IDLE				

Table E/5.1 Client Machine, Disconnected Regime

STATE / EVENT	CNECTED	REL-P	STR-P	MDF-P	MDR-P	
RELRQ	REL :REL-P					
STRRQ	STR :STR-P					
MDFRQ	MDF :MDF-P					
MDRRQ	MDR :MDR-P					
REL-R		RELCF +P-RELEASE :IDLE -:CNECTED				
STR-R			STRCF :TR-IDLE			
MDF-R				MDFCF :CNECTED		
MDR-R					MDRCF :CNECTED	
P-ABORT	PABIN :F1	PABIN :F1	PABIN :F1	PABIN :F1	PABIN :F1	
DISRQ	purge DIS :F1	purge DIS :F1	purge DIS :F1	purge DIS :F1	purge DIS :F1	
DIS	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	

Table E/5.2 Client Machine Connected Regime

STATE / EVENT	TR-IDLE	SEC-P	SECURE	SCM-P	RBK-P	COM-P
SECRQ	SEC :SEC-P					
SCMRQ	SCM :SCM-P			SCM :SCM-P		
RBKRQ	purge RBK :RBK-P resync	purge RBK :RBK-P resync	RBK :RBK-P			
SEC-R		SECCF + :SECURE - :CNECTED	SECCF :SECURE			
SCM-R				SCMCF :CNECTED		
RBK-R					RBKCF :CNECTED	
RBP	RBPIN :RBP-PQ	purge RBPIN :RBP-PQ		purge RBPIN :RBP-PQ		
COMRQ			COM :COM-P			
COM-R						COMCF :CNECTED
P-ABORT	PABIN :F2	PABIN :F2	PABIN :F3	PABIN :F5	PABIN :F4	PABIN :F4
DISRQ	DIS :F2	DIS :F2	DIS :F3	DIS :F5	DIS :F4	DIS :F4
DIS	DISIN :F2	DISIN :F2	DISIN :F3	DISIN :F5	DISIN :F4	DISIN :F4

Table E/5.3 (start) Client Machine - Transact and Secure Regimes

STATE / EVENT	TR-IDLE	RDL-P	APT-P	RDT-P	ETR-P ex Bulk Read	RBP-PQ
RDLRQ	RDL :RDL-P					
APTRQ	ckct=0 APT :APT-R					
RDTRQ	ckct=0 RDT :RDT-R					RBK :RBK-P
ETR-R					ETRCF :TR-IDLE	
SPR		SPRIN :RDL-P				
RDL-R		RDLCF :TR-IDLE				
APT-R			APTCF -:TR-IDLE + :SN. SENDING			
RDT-R				RDTCF - :TR-IDLE + :RV. RCVING		
RBKRQ		purge RBK :RBK-P				
RBP	see above	purge RBPIN :RBP-PQ resync	purge RBPIN :RBP-PQ resync	purge RBPIN :RBP-PQ resync		
P-ABORT	see above	PABIN :F2	PABIN :F16	PABIN :F6	PABIN :F7	PABIN :F4
DISRQ	see above	DIS :F2	DIS :F16	DIS :F6	DIS :F7	DIS :F4
DIS	see above	DISIN :F2	DISIN :F16	DISIN :F6	DISIN :F7	DISIN :F4

Table E/5.3 (end) Client Machine - Transact and Secure Regimes

STATE EVENT	SENDING	ERD-P	SN.ETR-P (append table)	ETR-PP	STATE EVENT	SN.ETR-P (append table)
DATRQ	if ckct< max DAT :SENDING				ETR-R	ETRCF :CL.TR- IDLE
CHKRQ	CHK inc ckct :SENDING					
CHK-R	dec ckct CHKIN :SENDING	dec ckct CHKIN :ERD-P	dec ckct CHKIN :ETR-P			
ERDRQ	if SV ERD :ERD-P					
ETRRQ	if CL ETR :SN.ETR-P					
ETR		ETRIN :ETR-PP				
ETRRP				ETR-R :TR-IDLE		
CAN	purge CANIN :CAN-PP	purge CANIN :CAN-PP	purge CANIN :CAN-PP			
RST	purge RSTIN :RST-PP	purge RSTIN :RST-PP	purge RSTIN :RST-PP			
P-ABORT	purge PABIN :F8	purge PABIN :F9	purge PABIN :F10	purge PABIN :F9		
DISRQ	purge DIS :F8	purge DIS :F9	purge DIS :F10	purge DIS :F9		
DIS	purge DISIN :F8	purge DISIN :F9	purge DISIN :F10	purge DISIN :F9		

Table E/5.4(start) Sender Machine Bulk Transfer

STATE / EVENT	SENDING	RST-P	RST-PP	CAN-P	CAN-PP	
CANRQ	purge CAN :CAN-P	purge CAN :CAN-P	purge CAN :CAN-P			
CAN-R				CANCF :TR-IDLE		
RSTRQ	purge RST :RST-P					
RST-R		RSTCF ckno rset :SENDING		:CAN-P		
RSTRP			RST-R ckno rset :SENDING			
CANRP					CAN-R :TR-IDLE	
CHK-R	*****	dec ckct :RST-P		:CAN-P		
CAN	*****	purge CANIN :CAN-PP	purge CANIN :CAN-PP	:CAN-P		
RST	*****	purge RSTIN :RST-PP		:CAN-P		
P-ABORT	*****	purge PABIN :F8	purge PABIN :F8	purge PABIN :F11	purge PABIN :CL.F2 or :SV.F2	
DISRQ	*****	purge DIS :F8	purge DIS :F8	purge DIS :F11	purge DIS :CL.F2 or :SV.F2	
DIS	*****	purge DISIN :F8	purge DISIN :F8	purge DISIN :F11	purge DISIN :CL.F2 or :SV.F2	

Table E/5.4(end) Sender Machine Bulk Transfer

STATE EVENT	F1	F1RCN-P	F1RESUME	F2	F2RCN-P	F2RCNPP
RCNRQ	P-Service RCN :F1RCN-P			P-Service RCN :F2RCN-P		
RCN-R		RCNCF - :F1 + :F1RESUME			RCNCF - :F2 + RBPIN : RBKPQ	
Previous Request			As for Connect Regime			
RCN				RCNIN :F2RCNPP	:F2RCN-P	
RCNRP						RCN-R - :F2 +:F2RESUME

Table E/5.5-1 Client Machine Fail States

STATE / EVENT	F2RESUME	F3	F3RCNPP	F4	F4RCN-P	F4RESUME
RCNRQ				P-Service RCN :F4RCN-P		
RCN-R					RCNCF - :F4 + :F4RESUME	
Previous Request						As for SECURE
RCN		RCNIN :F3RCNPP		RCNIN :F4RCN-PP	purge :F4RCN-P	
RCNRP			RCN-R - :F3 + :SECURE			
SEC-R	As SEC-P					
CAN	CAN-R RBPIN :RBKPQ					

STATE / EVENT	F4RCN-PP	F4SEC-P				
RCN-RP	RCN-R +:F4SEC-P - :F4					
SEC-R		SECCF + :SECURE - :CNECTED				

Table E/5.5-2 Client Machine Fail States

STATE / EVENT	F5	F5RCN-P	F5RESUME	F6	F6RCN-P	F6RESUME
RCNRQ	P-Service RCN :F5RCN-P			P-Service RCN :F6RCN-P		
RCN-R		RCNCF - :F5 + : F5RESUME			RCNCF - :F6 + :F6RESUME	
SCM			As for Transact Regime			
CANRQ						CAN RV.CAN-P

STATE / EVENT	F7	F7RCN-P	F7RESUME	F16	F16RCN-P	F16RCN-PP
RCNRQ	P-Service RCN :F7RCN-P			P-Service RCN :F16RCN-P		
RCN-R		RCNCF - :F7 +:F7RESUME			RCNCF -:F16 +:F16RES	
ETRRQ		ETR :CL.ETR-P				
RCN				RCNIN :F16RCN-PP		
RCNRP						RCN-R +:SENDING -:F16

Table E/5.5-3 Client Machine Fail States

STATE / EVENT	F8	F8RCN-P	F8RESUME	F8RCN-PP	F16RES	F9
RCNRQ	if CL P-Service RCN :F8RCN-P					
RCN-R		RCNCF - :F8 + : F8RESUME				
RSTRQ			RST :RST-P			
RCN	RCNIN :F8RCN-PP	purge RCNIN :F8RCN-PP				RCNIN :F9RCNPP
RCNRP				RCN-R - :F8 + :F8RESUME		
RST			purge RSTIN :RST-PP		:F16RES	
CANRQ					CAN :SN.CAN-P	
CAN			purge CANIN :CAN-PP			

Table E/5.6-1 Sender Machine Fail states

STATE / EVENT	F9RCNPP	F10	F10RCN-P	F10RCNPP	F10RESUME	
RCNRQ		if CL P-Service RCN :F10RCN-P				
RCN-R			RCNCF - :F10 + :F10RESUME			
RCN		RCNIN :F10RCNPP	purge RCNIN :F10RCNPP			
RCNRP	RCN-R - :F9 +:SN.ERD-P			RCN-R - :F10 +:SN.ETR-P		
RSTRQ						
ETRRQ					ETR :SN.ETR-P	

Table E/5.6-2 Sender Machine Fail States

STATE EVENT	F11	F11RCN-P	F11RESUME	F11RCN-PP		
RCNRQ	P-Service RCN :F11RCN-P					
RCN-R		RCNCF - :F11 + :F11RESUME				
CAN			CAN :CAN-P			
RCN	RCNIN :F11RCN-PP	purge RCNIN :F11RCN-PP				
RCNRP				- :F11 +&R.CL :TR-IDLE +&R.SV :F11RESUME		
RST			:F11RESUME			

Table E/5.6-3 Sender Machine Fail States

STATE / EVENT	IDLE	CON-PP				
CON	CONIN :CON-PP					
CONRP		CON-R +:CNECTED -ve :IDLE				
P-ABORT		DISIN :IDLE				
DIS		DISIN :IDLE				

Table E/5.7 Server Machine, Disconnected Regime

STATE / EVENT	CNECTED	REL-PP	STR-PP	MDF-PP	MDR-PP	
REL	RELIN :REL-PP					
STR	STRIN :STR-PP					
MDF	MDFIN :MDF-PP					
MDR	MDRIN :MDR-PP					
RELRP		REL-R +:IDLE -:CNECTED				
STRRP			STR-R :TR-IDLE			
MDFRP				MDF-R :CNECTED		
MDRRP					MDR-R :CNECTED	
SCM	SCM-R (repeat) :CNECTED					
P-ABORT	PABIN :F1	PABIN :F1	PABIN :F1	PABIN :F1	PABIN :F1	
DISRQ	purge DIS :F1	purge DIS :F1	purge DIS :F1	purge DIS :F1	purge DIS :F1	
DIS	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	purge DISIN :F1	

Table E/5.8 Server Machine, Connected Regime

STATE / EVENT	TR-IDLE	SEC-PP	SECURE	SCM-PP	RBK-PP	RBP-P
SEC	SECIN :SEC-PP					:RBP-P
SCM	SCMIN :SCM-PP			SCMIN :SCM-PP	:RBP-PP	:RBP-P
RBK	RBKIN :RBK-PP	purge RBKIN :RBK-PP	RBKIN :RBK-PP			RBKIN :RBK-PP
SECRP		SEC-R +:SECURE -:CNECTED	SEC-R :SECURE			
SCMRP				SCM-R :CNECTED		
RBKRP					RBK-R :CNECTED	
COM			COMIN :COM-PP			
CAN	CAN-R warning :TR-IDLE					
P-ABORT	PABIN :F2	PABIN :F2	PABIN :F3	PABIN :F4	PABIN :F3	PABIN :F2
DISRQ	DIS :F2	DIS :F2	DIS :F3	DIS :F4	DIS :F3	DIS :F2
DIS	DISIN :F2	DISIN :F2	DISIN :F3	DISIN :F4	DISIN :F3	DISIN :F2

Table E/5.9 (start) Server Machine, Transact and Secure Regimes

STATE / EVENT	TR-IDLE	RDL-PP	APT-PP	RDT-PP	COM-PP	RBP-P
RDL	RDLIN :RDL-PP					:RBP-P
APT	APTIN :APT-PP					:RBP-P
RDT	RDTIN :RDT-PP					:RBP-P
RDLRP		RDL-R :TR-IDLE				
SPRRQ		SPR :RDL-PP				
APTRP			APT-R +:RV. RVING -:TR-IDLE			
RDTRP				RDT-R +ve:SN. SENDING -:TR-IDLE		
COMRP					COM-R :CNECTED	
RBK		purge RBKIN :RBK-PP				As above page 134
RBPRQ	purge RBP :RBP-P	purge RBP :RBP-P	purge RBP :RBP-P	purge RBP :RBP-P		
P-ABORT	PABIN :F2	PABIN :F2	PABIN :F2	PABIN :F2	PABIN :F3	
DISRQ	DIS :F2	DIS :F2	DIS :F2	DIS :F2	DIS :F3	
DIS	DISIN :F2	DISIN :F2	DISIN :F2	DISIN :F2	DISIN :F3	

Table E/5.9 (end) Server Machine, Transact and Secure Regime

STATE / EVENT	RCVING	ERD-P	ETR-PP			
DAT	DATIN :RCVING					
CHK	CHKIN inc ckct :RCVING					
CHK-RP	dec ckct CHK-R :RCVING	dec ckct CHK-R :ERD-P	dec ckct CHK-R :ETR-PP			
ERD	if CL ERD-IN :ERD-P					
ETRRQ		ETR :CL.ETR-P				
ETR	if SV ETRIN :ETR-PP					
ETR-RP			ETR-R :SV. TR-IDLE			
CANRQ	purge CAN :CAN-P	purge CAN :CAN-P	purge CAN :CAN-P			
RSTRQ	purge RST :RST-P	purge RST :RST-P	purge RST :RST-P			
P-ABORT	PABIN :F8	PABIN :F9	PABIN :F10			
DISIN	DIS :F8	DIS :F9	DIS :F10			
DIS	DISIN :F8	DISIN :F9	DISIN :F10			

Table E/5.10 Receiver Machine Bulk Transfer

STATE / EVENT	RCVING	RST-P	RST-PP	CAN-P	CAN-PP	
CAN	purge CANIN :CAN-PP	purge CANIN :CAN-PP	purge CANIN :CAN-PP	purge CANIN :CAN-PP		
CANRP					CAN-R :TR-IDLE	
RST	RSTIN :RST-PP	:RST-P		:CAN-P		
RSTRP			RST-R :RCVING			
CAN-R				CANCF :TR-IDLE		
RST-R		RSTCF :RCVING		:CAN-P		
P-ABORT	PABIN :F8	PABIN :F8	PABIN :F8	PABIN :F11	PABIN :F8	
DISIN	DIS :F8	DIS :F8	DIS :F8	DIS :F11	DIS :F8	
DIS	DISIN :F8	DISIN :F8	DISIN :F8	DISIN :F11	DISIN :F8	

Table E/5.10 (end) Receiver Machine Bulk Transfer

STATE / EVENT	F1	F1RCN-PP	F1RESUME = CNECTED plus	F2	F2RCN-PP	F2RESUME
RCN	RCNIN :F1RCN-PP			RCNIN :F2RCN-PP		
RCNRP		RCN-R +:F1RESUME -:F1			RCN-R +:F2RESUME - :F2	
RBK			RBK-R (warning) :CNECTED			RBKIN :RBK-PP
COM			COM-R (warning) :CNECTED			
STR						purge tran STRIN :STR-PP
ETR						ETR-R warning :TR-IDLE
SCM			SCM-R (as previous) :CNECTED			SCMIN :SCM-PP
CAN						CAN-R :TR-IDLE

Table E/5.11 (start) Server Machine Fail States

STATE / EVENT	F3	F3RCN-PP	F3RCN-P	F4	F4RCN-PP	F4RESUME
RCN	RCNIN :F3RCN-PP		purge RCNIN :F3RCN-PP	RCNIN :F4RCN-PP		
RCNRP		RCN-R - :F3 +:SECURE			RCN-R - :F4 +:F4RESUME	
RCNRQ	P-Service RCN :F3RCN-P					
RCN-R			RCNCF - :F3 +:SECURE			
SCM						SCMIN :F4SCM-PP

Table E/5.11 (cont.) Server Machine Fail States

STATE / EVENT	F8	F8RCN-P	F8RCN-PP	F8RESUME	F9	F9RCN-P
RCNRQ	P-Service RCN :F8RCN-P				P-Service RCN :F9RCN-P	
RCN-R		RCNCF -:F8 +:F8RESUME				RCNCF - :F9 if CL:ERD-P if SV :ETR-PP
RCN	RCNIN :F8RCN-PP	:F8RCN-P				
RCNRP			RCN-R - :F8 +:F8RESUME			
RSTRQ				RST :RST-P		
RST				RSTIN :RST-PP		
CAN				CANIN :CAN-PP		
ETR				ETR-IN :ETR-PP		

Table E/5.12 (start) Receiver Machine Fail States

STATE EVENT	F10	F10RCN-P	F10RCN-PP	F10RES	F11	F11RCN-P
RCNRQ	P-Service RCN :F10RCN-P				P-Service RCN :F11RCN-P	
RCN-R		RCNCF -:F10 +:ETRIN ETR-PP				RCNCF -:F11 +:F11RES
RCN	RCNIN :F10RCN-PP	:F10RCN-P			RCNIN :F11RCN-PP	if R.SN :F11RCN-P if R.CL purge RCNIN :F11RCN-PP
RCNRP			RCN-R -:F10 +:F10RES			
ETR				ETRIN :ETR-P		

Table E/5.12 (cont) Receiver Machine Fail States

STATE EVENT	F11RCN-PP	F11RES				
RCNRP	RCN-R -:F11 +&R.SN :F11RES +&R.CL :TR-IDLE					
CANRQ		CAN :CAN-P				
CAN		:F11RES				
RST		:F11RES				

Table E/5.12 (end) Receiver Machine Fail States

EVENT \ STATE	Fi $1 \leq i \leq 16$	FiRCN-P	FiRCN-PP	FiRESUME FiRES		
P-ABORT		PABIN :Fi	PABIN :Fi	PABIN :Fi		
DISRQ		purge DIS :Fi	purge DIS :Fi	purge DIS :Fi		
DIS		purge DISIN :Fi	purge DISIN :Fi	purge DISIN :Fi		

Table E/5.13 All Machines, Failure During Recovery

E.7 GROUPING

Table E/6 shows the permissible seqences of service requests within groups. These sequences are only permissible where they would be valid with no grouping.

PREVIOUS \ NEXT	RDL temporary table	MDF	MDR	RDL	SEC	EGR
BGG	Y	Y	Y	Y		NS
RDL temporary table	Y	Y	Y			Y
MDF	Y	Y	Y			Y
MDR	Y	Y	Y			Y
RDL				Y	Y	Y
SEC						Y

Table E/6 Permissible sequences within groups

APPENDIX F

DIAGNOSTIC REASON CODES

F.1 INTRODUCTION

Diagnostic reason codes define the presumed cause of an error, The codes are classified into related groups and numbered accordingly.

In the tables below the Type column lists the possible Severity Codes associated with the errors. The meaning of the severity codes is:

1 = Warning

2 = Recoverable Error

3 = Uncorrectable Error

The groupings of error codes used below are:

- General RDA Diagnostics;
- Protocol and Supporting Service Related Diagnostics;
- Association Related Diagnostics;
- Data Definition and Manipulation Management;
- Transaction Management;
- Bulk Data Transfer;
- Recovery Related Diagnostics.

F.1.1 General RDA Diagnostics

Type	Identifier	Meaning
23	0	No Reason
123	1	Server Error (unspecific)
23	2	System Shutdown
123	3	Server Management Problem
1 3	4	Bad Account
1 3	5	Security not passed
1	6	Expect delay
123	7	Client error (unspecific)
123	8	Subsequent error

F.1.2 Protocol and Supporting Service Related Diagnostics

Type	Identifier	Meaning
3	1000	Conflicting parameter values
3	1001	Unsupported parameter values
3	1002	Mandatory parameter not set
3	1003	Unsupported parameter
3	1004	Duplicated parameter
3	1005	Illegal parameter type
3	1006	Unsupported parameter type
3	1007	RDA protocol error (unspecific)
3	1008	RDA protocol - procedure error
3	1009	RDA protocol - facility not negotiated
3	1010	RDA protocol - message error
3	1011	Lower layer failure
23	1012	Lower layer addressing error
23	1013	Time out
23	1014	System shutdown

F.1.3 Association Related Diagnostics

Type	Identifier	Meaning
3	2000	Association with user not allowed
3	2001	Unsupported Quality of R-Service
3	2002	Unsupported Class of R-Service
3	2003	Bad Association Id
1 3	2007	Bad current account
3	2010	Suspend not supported
3	2011	Server transaction secure

F.1.4 Data Definition and Manipulation Management;

Type	Identifier	Meaning
1 3	3000	Macro name exists
1	3001	Macro does not exist
3	3002	Defective macro body

F.1.5 Transaction Management

Type	Identifier	Meaning
123	4000	Unspecific database error
2	4001	Deadlock detected
23	4002	Subordinate process failure

F.1.6 Bulk Data Transfer

Type	Identifier	Meaning
3	5000	Invalid table name
3	5001	No access
2	5002	Data lost
2	5003	Checkpoint unsecured
3	5004	Invalid data
1 3	5005	Outside Checkpoint Window

F.1.7 Recovery Related Diagnostics

Type	Identifier	Meaning
3	6000	Invalid association parameters
3	6001	Association id unknown
3	6002	Association complete
2	6003	Awaiting database recovery
	6004	Association not recoverable