INTERNATIONAL STANDARD

ISO/IEC 9579

Second edition
2000-02-15

Information technology — Remote database access for SQL with security enhancement

Technologies de l'information — Accès à la base de données à distance pour SQL avec sécurité accrue

Reference number
ISO/IEC 9579:2000(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

Tables

Figures

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 9579 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This second edition cancels and replaces the first edition (ISO 9579:1999), which has been technically revised.

Annexes A to E and G form a normative part of this International Standard. Annexes F, H and I are for information only.

Introduction

Remote Database Access for SQL (RDA/SQL) International Standard is a member of a set of International Standards produced to facilitate the interworking of computer systems. This International Standard conforms to the model defined in ISO/IEC 10032, *Information technology – Reference Model of Data Management*.

Remote Database Access for SQL can be used to provide remote data access to a database management system conforming to ISO/IEC 9075 (Database Language SQL).

The goal of Remote Database Access for SQL is to allow, with a minimum of technical agreement outside this International Standard, the interconnection of applications and database systems:

- from different manufacturers,
- under different managements,
- of different levels of complexity,
- exploiting different technologies.

An application may itself be a database system and therefore this International Standard can be used to support multi-database system interworking.

Information technology — Remote database access for SQL with security enhancement

1 Scope

This International Standard, Remote Database Access for SQL (RDA), defines a model for the remote interaction of an SQL-client and one or more SQL-servers through communication media, and defines the encoding of messages, the semantics of messages and associated facilities for mediating the interaction between one SQL-client and one SQL-server.

This International Standard also defines a mapping of the RDA Protocol to the specific communication infrastructures TCP/IP and Transport Layer Security (TLS).

This International Standard relies upon the facilities provided by ISO/IEC 9075 (SQL) and ISO/IEC 9075-3 (SQL/CLI).

This International Standard also:

- identifies potential security vulnerabilities in remote database access using RDA,
- defines RDA facilities which protect against the potential vulnerabilities.

Normative annexes provide:

- a Conformance Proforma,
- an optional language independent Application Programming Interface defined in the notational conventions of ISO/IEC 9075-3 (SQL/CLI) for invoking RDA Operations,
- an optional mapping of ISO/IEC 9075-3 (SQL/CLI) functions to RDA Operations,
- definitions of optional SQL-servers, the RDA Location Server and the RDA Support Server, to facilitate interoperation and data distribution in a heterogeneous environment,
- a set of security profiles that identify which RDA facilities and other security facilities are required for different levels of protection against potential vulnerabilities.

Informative annexes provide:

- an analysis of security service requirements,
- an ASN.1 specification for the RDA Protocol,
- an ASN.1 specification for the encoding of multiple rows.

This International Standard does not constrain:

- conforming RDA-client environments to be implemented using any particular processor decomposition,
- conforming RDA-server environments to be implemented using any particular processor decomposition.

This International Standard refers to but does not define:

- protocols and security mechanisms for communication confidentiality, integrity and authentication of communicating peers,
- digital signature and authentication mechanisms supported by protocol elements of RDA.

This International Standard does not define:

- algorithms for query decomposition or for the combining of results in a distributed database environment,
- mechanisms for recovery in the event that transaction co-ordination fails,
- mechanisms for storage integrity and confidentiality using cryptography,
- mechanisms to counter Denial of Service attacks.

2 Remote Database Access for SQL (RDA/SQL)

2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

2.1 International Standards

ISO/IEC 8824-1:1995	*Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*
ISO/IEC 8825-1:1995	*Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*
ISO/IEC 8825-2:1996	*Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*
ISO/IEC 7498-2:1989	*Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*
ISO/IEC 9075:1992	*Information technology – Database Languages SQL*
ISO/IEC 9075-3:1995	*Information technology – Database Languages SQL – Part 3: Call Level Interface*
ISO/IEC 9075-4:1996	*Information technology – Database Languages SQL – Part 4: Persistent Stored Modules*
ISO/IEC 9594-8:1995 |	ITU-T Recommendation X.509 *Information technology – Open Systems Interconnection – The Directory: Authentication Framework.*
ISO/IEC 10032:1995	*Information technology – Reference Model of Data Management*
ISO/IEC 10646-1:1993	*Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

2.2 Internet Engineering Task Force

RFC 791	*Internet Protocol*
RFC 793	*Transmission Control Protocol*
RFC 819	*The Domain Naming Convention for Internet User Applications*
RFC 1122	*Requirements for Internet Hosts – Communication Layers*
RFC 1123	*Requirements for Internet Hosts – Application and Support*
RFC 2246	*The TLS Protocol*

Internet Engineering Task Force standards may be obtained in electronic form from the InterNIC Directory and Database Services at *http://www.internic.net* and *ftp://ftp.internic.net*.

2.3 Institute of Electrical and Electronics Engineers

IEEE 754-1985 *Standard for Binary Floating-Point Arithmetic*

Institute of Electrical and Electronic Engineers (IEEE) standards may be obtained from *IEEE Customer Service, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA* or ordered electronically from *http://www.ieee.org*.

3 Interoperability

This clause defines the compatibility of RDA-client or RDA-server implementations of the Operations and Protocol defined by this edition of this International Standard with RDA-server or RDA-client implementations respectively of the Operations and Protocol defined by this and other editions of this International Standard.

3.1 Interoperability between implementations

An RDA-client or RDA-server implementation that conforms to this International Standard will interoperate with any other RDA-server or RDA-client that conforms to this International Standard provided that the implementations use the same Transport Mapping.

Any cryptographic algorithms and authentication mechanisms must be common to both the RDA-client and RDA-server if they are to interoperate.

NOTE 1 – Cryptographic algorithms are not defined by this International Standard.

NOTE 2 – There are circumstances under which the RDA Protocol and RDA Operations defined by this International Standard require that RDA Messages between interoperating implementations are rejected. These circumstances include the failure to satisfy authentication requirements and the use by one implementation of an encoding that is not supported by the other (the Default Encoding is always supported).

3.2 Interoperability with conforming OSI implementations

An earlier (three part) version of this International Standard published in 1997, now seperceded, maps the Service and Protocol to an OSI transport provider. This edition of this International Standard maps the RDA Protocol to a TCP/IP transport provider. Implementations conforming to the obsolete International Standard can therefore co-exist with implementations conforming to this International Standard but cannot directly interoperate.

3.3 Interoperability with future editions

Features have been included in this version of the Protocol to permit server implementations to detect which version of the protocol a client has implemented and to behave appropriately.

Future editions of this International Standard will be compatible with this edition to the extent that:

- RDA Operations and Protocol defined by this edition will be retained in future editions using the same encodings.
- Changes to the Operations, Protocols and encodings in future editions will be extensions that are recognised by implementations of this edition and discarded after raising an exception.

4 Definitions, Conventions and Notations

4.1 Definitions

For the purposes of this International Standard, the definitions given in ISO/IEC 9075 (SQL) and ISO/IEC 9075-3 (SQL/CLI) and the following definitions apply.

The following terms are defined in RFC 791, RFC 793, RFC 819, RFC 1122, RFC 1123:

IP Address,

Local Socket,

Foreign Socket,

Port Number,

Push.

The following terms are defined in ISO/IEC 7498-2:

Access Control,

Authentication,

Confidentiality,

Integrity,

Non-repudiation.

The following terms are defined in ISO/IEC 9594-8:1997 | ITU-T X.509 (1997):

Attribute Certificate,

Public Key Certificate.

The following term is defined in ISO/IEC 10032:

Distribution Controller.

In addition, the following definitions apply:

RDA Message: a protocol element as defined by this International Standard exchanged between an RDA-client and an RDA-server.

RDA Operation: a facility that is accessible remotely provided by an RDA-server, together with the means of invoking the facility through a protocol, the encoding of parameters that influence or result from an invocation of the facility, and rules governing the invocation of the facility.

RDA Programming Interface: a language independent Application Programming Interface defined in the notational conventions of ISO/IEC 9075-3 (SQL/CLI) for invoking RDA Operations.

RDA Protocol: the set of permissible exchanges of requests and responses between an RDA-client and an RDA-server, together with the encoding of the exchange, and rules governing the exchange.

RDA Relay: a system that receives RDA protocol elements as defined in this International Standard for the purpose of forwarding them to the intended recipient.

4.2 Conventions

4.2.1 Convention for Figures

The convention used for figures is that defined in the Reference Model of Data Management, ISO/IEC 10032.

4.2.2 Naming of Concepts

Where a concept has been defined in ISO/IEC 9075 (SQL) and used in this International Standard, the name used in ISO/IEC 9075 (SQL) is used for that concept.

Concepts whose name begins with 'RDA-' are defined in this International Standard.

4.2.3 Naming of Parameters

The same identifier is used to designate both an RDA/API routine parameter and an RDA Protocol or Operation parameter where the items are semantically equivalent.

> NOTE 3 – For data values of type other than RDAOctetString the encoding of this value may differ between the parameter in the RDA-client environment, the parameter in the protocol and the parameter in the RDA-server environment. This may be due to, for example, architecture, precision or language differences between the client and the server system.

Where there is a parameter in an SQL/CLI function which corresponds to an RDA Operation parameter or an RDA/API parameter, and the SQL/CLI parameter is semantically equivalent to the RDA parameter, the SQL/CLI parameter name is used.

Whether an identifier designates an SQL/CLI parameter, RDA/API client parameter, protocol parameter or RDA/API parameter is determined by the context.

4.2.4 Specification of RDA Protocol, RDA Operations and RDA encoding elements

The definition of each RDA protocol element, RDA Operation element or Encoding component has up to four parts

- *function*, a short statement of the purpose of the element,
- *encoding*, the syntax for the encoding of the element in terms of RDA Protocol or RDA Operation parameters,
- *parameters*, a definition of the RDA Protocol parameters or RDA Operation parameters for the element,
- *rules*, a sequence of Protocol Rules or Operation Rules for the element to be evaluated within the RDA-client environment or the RDA-server environment as indicated.

4.2.5 Evaluation of Rules

Rules are used:

- to specify RDA Protocol Rules,
- to specify RDA Operation Rules.

When Rules are evaluated, the required effect is that which would be obtained by beginning with the first Rule and evaluating the rules in numerical sequence until a rule is evaluated that specifies or implies either a change in sequence or termination of the evaluation of the Rules. Unless otherwise specified or implied by a specific Rule that is evaluated, evaluation of Rules terminates when the last in a sequence has been evaluated.

Evaluation of a sequence of Rules *R1* may involve the evaluation of another sequence of Rules *R2* referred to in *R1*. Termination of *R2* does not of itself terminate the evaluation of *R1*.

If the evaluation of a sequence of Rules causes an exception to be raised, evaluation of the sequence of Rules causing the exception is terminated after raising the exception.

If the evaluation of a Rule *R* results in the evaluation of a sequence of General Rules of ISO/IEC 9075-3 (SQL/CLI) then that sequence of General Rules is evaluated in the manner defined by ISO/IEC 9075-3 (SQL/CLI). If such evaluation causes an exception to be raised, then the exception is deemed to have been raised by *R*.

4.3 Notations

4.3.1 SQL/CLI functions

An SQL/CLI function *X* defined in ISO/IEC 9075-3 (SQL/CLI) is referred to in this International Standard as SQL*X*. This notation makes no assumption as to whether the function is a “by value” function or a “by reference” function.

4.3.2 Implicit encoding definitions

For each definition of an encoding with identifier *X*, an additional definition is implied for an encoding *X*List defined as SEQUENCE OF *X*.

4.3.3 Encoding Attributes

For the purposes of this document, each encoded item, whether basic or compound, is deemed to have the following attributes which can be derived from the encoding or from the RDA encoding syntax:

Encoded Length: the length in octets of an encoding, including any octets that specify length.

Encoded Type: the type of an encoding.

Encoded Value: the value encoded.

Furthermore, certain encoded items have the following additional attributes that can be derived from the encoding:

Encoded Character Length: the length in characters of an encoding of a value of type RDACharString.

Encoded Count: the number of items in a SEQUENCE OF encoding.

4.3.4 Notation for encoding syntax

The notation for defining encoding syntax is ASN.1 Basic Notation as defined in ISO/IEC 8824-1.

5 Model and Facilities

5.1 Model

ISO/IEC 9075 (SQL) defines an SQL-environment, SQL-client and SQL-server. This International Standard (RDA) defines how an SQL-client and SQL-servers interact when supported by an underlying communication system. Concrete definitions for some aspects of the SQL model and some SQL concepts are given to the extent that is required for remote, heterogeneous, interoperation of an SQL-client and an SQL-server using the facilities defined by this International Standard, RDA.

The model adds the following key concepts: *RDA-client, RDA-server*, *Transport Provider,* and *Transport Connection.* Subsequent subclauses of this clause define each of these concepts.

Figure 1 provides a representation of the interrelationship of these components and related SQL terms.

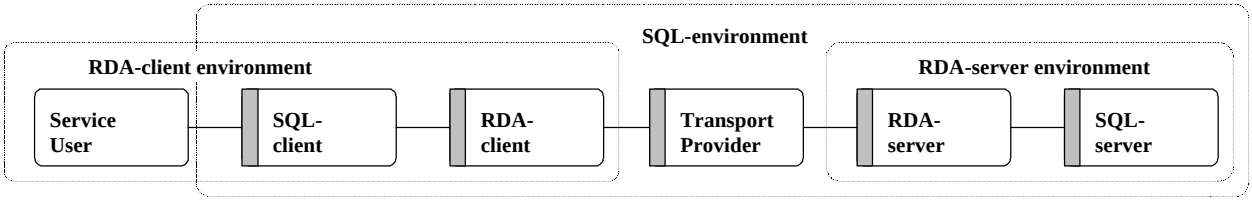

Figure 1 – RDA model of SQL-environment

The model adopted by this International Standard is that an RDA-client environment, which includes a Service User and an SQL-client, uses one or more Transport Providers to connect to RDA-server environments, each of which has one or more SQL-servers associated with it thereby establishing SQL-sessions between the SQL-client and the SQL-servers. Each SQL-session is associated with an SQL-connection. These are in turn supported by Transport Connections within Transport Providers that are established and terminated as required for communicating through the SQL-connection. More than one Transport Provider may be used concurrently to support concurrent SQL-sessions. A particular SQL-server may be in more than one RDA-server environment so the particular Transport Provider used to support a session between an SQL-client and a particular SQL-server may differ between SQL-sessions.

5.2 The RDA-client environment

The RDA-client environment is described with reference to five functional components: a *Service User*, *SQL-client Services* which establish connections with SQL-servers and co-ordinate operations involving more than one SQL-server, *RDA-client Services* which provide the client end of the RDA Protocol and the RDA Operations, a *Transport Mapping* that interfaces to a communications infrastructure and an optional *RDA Location Server* that facilitates control of distributed data. How these five functional components interrelate is depicted in Figure 2.

NOTE 4 – An implementation containing a conforming RDA-client need not materialise the five different components corresponding to these five functional components nor their intermediate interfaces.

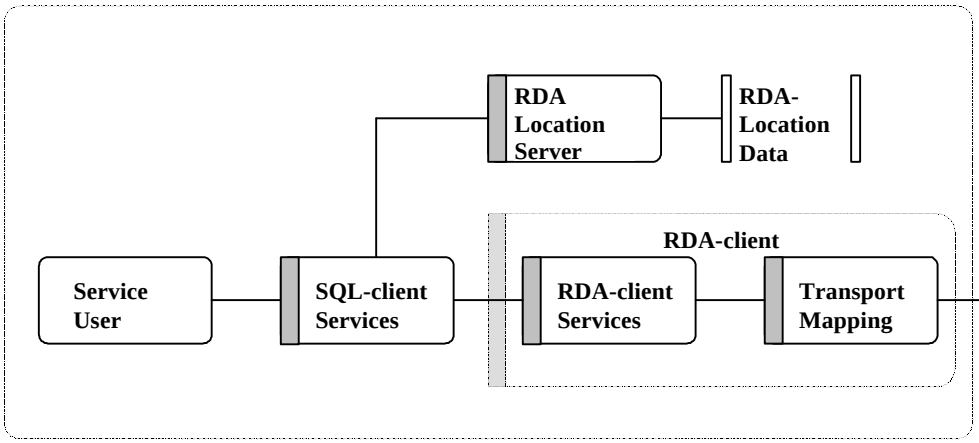

Figure 2 – Model of the RDA-client environment

5.2.1 Service User

The Service User is the component of the RDA-client environment that invokes SQL/CLI routines.

NOTE 5 – The Service User is not further defined by this International Standard.

5.2.2 SQL-client Services

SQL-client Services:

- map the SQL/CLI routines used within the RDA-client environment by the Service User to the RDA Operations defined by this International Standard,
- hold SQL Diagnostic and SQL Descriptor information,
- buffer the results of SQL operations,
- perform locally those SQL operations that can be carried out without communication with an RDA-server environment,
- co-ordinate SQL operations involving more than one SQL-server.

SQL-client Services also co-ordinate SQL operations involving more than one SQL-server by:

- processing the SQL-connection statements, that is <connect statement>, <set connection statement> and <disconnect statement>,
- processing the SQL-transaction statements, that is <set transaction statement>, <set constraints mode statement>, <commit statement> and <rollback statement>,
- co-ordinating the protocol flows for multiphase commitment,
- mapping instances of entities that have only local validity into instances of entities with more extended validity
- translating an SQL statement into a number of other SQL statements, causing these statements to be executed, and combining the results of their execution.

NOTE 6 – Components outside the RDA-client environment may include further facilities for co-ordinating SQL operations involving more than one SQL-server.

Those constraints on SQL-client Services that must be enforced to ensure conforming behaviour of the SQL-environment when RDA components are present in that environment are defined in Annex C.

NOTE 7 – SQL-client Services are not further defined by this International Standard.

5.2.3 RDA-client Services

The RDA-client Services are the client component of the RDA Protocol and RDA Operations. RDA-client Services use the RDA Protocol defined in clause 6 to communicate RDA-server Services through a Transport Provider.

RDA-client Services:

- allocate Connection Idents, Request Idents and Statement Idents,
- establish and terminate Transport Connections,
- use Transport Facilities to send request messages to an RDA-server and receive response messages from that RDA-server,
- link response messages to the corresponding request messages and detect protocol errors.

RDA-client Services are defined as part of the definition of the RDA Protocol in clause 6 and RDA Operations in clause 7.

5.2.4 Transport Mapping

A Transport Mapping provides a mapping from the RDA Protocol to a Transport Provider, mapping the RDA Operations to a specific transport protocol and providing the facilities for communicating between the transport addresses of RDA-clients and RDA-servers.

Transport Mappings are defined in clause 10.

5.2.5 RDA-client

An RDA-client is RDA-client Services together with a Transport Mapping.

5.2.6 RDA Location Server

The RDA Location Server is an optional facility to assist in the distribution of data in a heterogeneous environment by providing a standard mechanism for identifying the location of SQL-servers. The RDA Location Server is defined in Annex D.

5.3 The RDA-server environment

The RDA-server environment is described with reference to three functional components: a *Transport Mapping* which interfaces to the communications infrastructure, *RDA-server Services* which provide the server end of the RDA Operations and Protocol and *SQL-server*. How these three functional components interrelate is depicted in Figure 3.

NOTE 8 – An implementation containing a conforming RDA-server need not materialise three different components corresponding to these three functional components nor their intermediate interfaces.

Associated with each RDA-server environment there is an optional *RDA Support Server*.

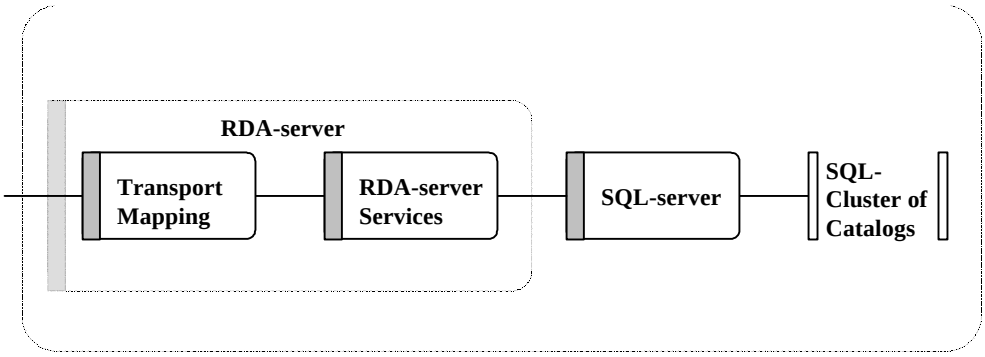

Figure 3 – Model of the RDA server environment

5.3.1 Transport Mapping

A Transport Mapping provides a mapping from a specific Transport Provider to the RDA Protocol.

Transport Mappings are defined in clause 10.

5.3.2 RDA-server Services

The RDA-server Services are the server component of the RDA Protocol and RDA Operations. RDA-server Services are invoked by RDA-client Services by means of the RDA Protocol defined in clause 5.6 from precisely one instance of RDA-client Services through a Transport Provider.

RDA-server Services:

- check those aspects of an RDA Request that are not checked by an SQL-server or its interface,
- translate communication concepts to and from local concepts, and
- invoke SQL-server facilities.

RDA-server Services are defined as part of the definition of the RDA Protocol in clause 6 and RDA Operations in clause 7.

5.3.3 RDA-server

An RDA-server is RDA-server Services together with a Transport Mapping, and maps the Operations and Services defined by this International Standard to the SQL-binding offered by the SQL-server.

5.3.4 SQL-server

An SQL-server functional component is defined in ISO/IEC 9075 (SQL).

An SQL-server has associated with it a portion of SQL-data that is described by a set of Catalogs termed a “Cluster of Catalogs” by ISO/IEC 9075 (SQL).

5.3.5 RDA Support Server

The RDA Support Server is an optional facility to assist interoperation in a heterogeneous environment by providing a standard mechanism for identifying characteristics of SQL-servers retrieving historic operation information. The RDA Support Server is defined in Annex E.

5.4 RDA concepts and the mapping of SQL/CLI concepts

5.4.1 Application Communication Areas

The communication between a Service User and an SQL-environment is facilitated by means of *Application Communication Areas*. These are Attributes, diagnostic areas and descriptor areas and are described in the following subclauses.

Although many functional components contribute to Application Communication Areas, this International Standard locates these areas in the RDA-client environment and designates the SQL-client Services as responsible for the direct maintenance of these areas in the RDA-client environment.

The RDA Protocol and RDA Operations provide facilities to ensure that the contents of the RDA-client environment communication areas and copies of these areas within RDA-server environments are synchronised to the extent required for conforming behaviour of the SQL-environment.

5.4.1.1 Attributes

ISO/IEC 9075-3 (SQL/CLI) uses the concept of *environment attribute* to identify certain characteristics of an SQL-environment, *connection attribute* to identify certain characteristics of an SQL-connection and *statement attribute* to identify certain characteristics of an SQL-statement. RDA regards these characteristics collectively as *Attributes*.

The RDA-client environment and the RDA-server environment must have a consistent view of Attributes. An operation RDAClientAttribute is provided which can be used to synchronise the RDA-server environment copy of Attribute values with the corresponding values within an RDA-client environment. In addition, each response message includes a parameter that permits an RDA-client environment to synchronise its copy of the Attribute values with the corresponding values within an RDA-server environment.

5.4.1.2 Diagnostics areas

ISO/IEC 9075 (SQL) defines SQL diagnostics areas and ISO/IEC 9075-3 (SQL/CLI) specialises the definition to CLI diagnostics areas.

Information for inclusion in diagnostics areas can be generated by the SQL-client Services, the RDA-client Services, the Transport Provider, the RDA-server Services or the SQL-server.

Each response message includes a parameter that communicates to the RDA-client environment diagnostics generated within the RDA-server environment.

5.4.1.3 Descriptor areas

ISO/IEC 9075 (SQL) defines SQL descriptor areas and ISO/IEC 9075-3 (SQL/CLI) specialises the definition to CLI descriptor areas. Associated with each allocated SQL-statement there are four specific descriptor areas: Implementation Parameter Descriptor *IPD*, Implementation Row Descriptor *IRD*, Application Parameter Descriptor *APD* and Application Row Descriptor *ARD*.

Descriptor information is communicated with the results of a request to prepare an SQL-statement, with the request to execute an SQL-statement and with the results of a request for rows of results.

5.4.2 SQL_TEXT

ISO/IEC 9075 (SQL) defines a character set SQL-TEXT whose character repertoire contains every character used in the SQL language and every character that is in any other character set supported by the SQL-implementation. To ensure heterogeneous interoperation, this International Standard requires the character repertoire of SQL_TEXT to be the Basic Multilingual Plane of ISO/IEC 10646.

5.4.3 SQL-session and SQL-connection

ISO/IEC 9075 (SQL) defines an SQL-connection associated with each SQL-session. An SQL-connection is supported by an RDA-client and an RDA-server pair. An RDA-client and an RDA-server can each support at most one SQL-connection at any time.

A Transport connection (as defined in 5.5.4) is established between the RDA-client and the RDA-server of an SQL-connection as required for communication. Each Transport Connection is therefore associated with at most one SQL-connection at any time.

The underlying Transport Connection of an SQL-connection may be terminated and re-established without disruption of the SQL-connection.

5.4.4 SQL User Name and Password

ISO/IEC 9075 (SQL) restricts access to SQL-data on the basis of Privileges assigned to Users.

ISO/IEC 9075-3 (SQL/CLI) provides for Users to identify themselves by means of a User Name which may be authenticated by a Password. The SQL User Name and Password are used by this International Standard as a basis for authenticating the identity of the User.

Additionally, the User Name may be authenticated by other types of RDA Sevice User Authentication as described in 5.8.3.

> NOTE 9 – The User Name does not necessarily identify the user uniquely but is sufficient for an SQL-server to allocate the user with appropriate access privileges.

5.4.5 Multi-site Transactions

ISO/IEC 9075 (SQL) permits SQL-transactions to involve more than one SQL-server. Each SQL-connection is mapped to a separate RDA-client, RDA-server pair. A COMMIT or ROLLBACK request from a Service User is mapped onto a series of transaction co-ordination exchanges with each participating RDA-server unless the SQL-transaction is part of an Encompassing Transaction, in which case the request is handled outside the protocol defined by this International Standard.

5.4.6 SQL/CLI Handles

ISO/IEC 9075-3 (SQL/CLI) uses the concept of a *handle* to uniquely identify an allocated resource. In an RDA environment the allocated resource has two sub-components, one in the RDA-client environment and one in the RDA-server environment. A handle is not required to have validity outside the compilation unit that allocated it. In an RDA environment, therefore, an allocated resource may have two handles associated with it: one with validity within the RDA-client environment to identify the RDA-client sub-components of the resource, another within the RDA-server environment to identify the RDA-server sub-components of the allocated resource.

5.4.7 Connection Ident

Each SQL-connection is assigned a Connection Ident by the RDA-client Services. This Connection Ident is a code that differs from the Connection Ident of any other allocated SQL-connection within the RDA-client environment. A Connection Ident thus serves to establish a correspondence between a Connection Handle within an RDA-client environment and a Connection handle within an RDA-server environment.

5.4.8 Statement Ident

Each allocated SQL-statement is assigned a Statement Ident by the RDA-client Services. This Statement Ident is a code that differs from the Statement Ident of any other allocated SQL-statement within the SQL-session. A Statement Ident thus serves to establish a correspondence between a Statement Handle within an RDA-client environment and a Statement handle within an RDA-server environment.

5.4.9 Request Ident

Each request for an RDA Operation is assigned a Request Ident by the RDA-client Services. This Request Ident is a code that differs from the Request Ident of any other RDA Operation request originating from that RDA-client environment.

RDA-client Services use the Request Ident to identify the request messages for an RDA Operation to which a response message relates.

5.4.10 Encodings

RDA protocol elements and their constituent parts are encoded for transmission. Encodings are defined in clause 9. For each Transport Mapping there is one encoding, the *Default Encoding* which every conforming implementation supporting that Transport Mapping must provide.

5.5 RDA Model of Transport

5.5.1 Transport Provider

A *Transport Provider* is a communications infrastructure that permits the RDA-client and RDA-server components in an SQL-environment to communicate using the operations and the protocol defined by this International Standard.

A Transport Provider is required to be reliable and always available to RDA. The format of RDA protocol elements includes a length field to assist mapping to streaming transport providers.

Mappings of the RDA Operations and the RDA Protocol to Transport Providers are defined in clause 10.

NOTE 10 – An SQL-client may invoke RDA Operations through more than one RDA-client and those RDA-clients need not all use the same Transport Provider.

NOTE 11 – Transport Providers are not defined by this International Standard.

5.5.2 Transport Address

Communication takes place between an RDA-client and an RDA-server each of which is identified within a Transport Provider by one or more *Transport Addresses*.

5.5.3 Destination SQL-server Name

An SQL-server associated with an RDA-server has a *Destination SQL-server Name* which is different from the Destination SQL-server Name of every other SQL-server associated with that RDA-server.

NOTE 12 – The Destination SQL-server Name of an SQL-server may be the <SQL-server name> by which it is identified to SQL-clients "local" to the server in a <connect statement> and may be the same as the Destination SQL-server Name of an SQL-server associated with some other RDA-server.

More than one SQL-server may be associated with an RDA-server and hence with a Transport Address within a Transport Provider. Also, one SQL-server may be associated with more than one RDA-server.

In an RDA-client environment an SQL-server name presented to an SQL-client is mapped by the SQL-client Services to a Transport Provider, a Transport Address and a Destination SQL-server Name.

NOTE 13 – The three values Transport Provider, Transport Address and Destination SQL-server Name when considered together by an SQL-client uniquely identify an SQL-server from among the SQL-servers that it is possible for that SQL-client to connect to using the available Transport Providers.

5.5.4 Transport Connection

A Transport Connection is a service provided by a Transport Provider for communicating encodings between an RDA-client and an RDA-server with the minimum of protocol overhead.

5.5.5 Transport Facilities

Each Transport Provider must have five Transport Facilities available at each associated RDA-client or RDA-server:

Transport Connect: a facility that requests the establishing of a Transport Connection.

Transport Disconnect: a facility that requests the termination of the Transport Connection. This facility shall not disrupt encodings that have already been sent through the Transport Connection.

Transport Fail: a facility that terminates the Transport Connection, possibly with loss of RDA Messages.

Transport Send: a facility that sends an encoding through a Transport Connection.

Transport Receive: a facility that receives an encoding through a Transport Connection.

NOTE 14 – Transport Facilities are not further defined by this International Standard.

5.6 RDA Facilities for Transport Connections

The RDA Transport Connection facilities are:

RDA Suspend and Resume: a mechanism for terminating and re-establishing a Transport Connection without disrupting the associated SQL-connection.

RDA Encoding: a mechanism for encoding RDA protocol elements and their constituent parts for sending through a Transport Connection.

The following subclauses describe how features defined by this International Standard are used to provide the RDA Transport Connection facilities.

5.6.1 RDA Suspend and Resume Facility

The underlying Transport Connection of an SQL-connection may be terminated and re-established without disruption of the SQL-connection, as defined in 5.4.3. Attributes are provided to control this behaviour:

CLIENT RESUME PERMITTED: Indicates whether an RDA-server permits an RDA-client to terminate and re-establish Transport Connections within an SQL-connection.

SERVER RESUME SUPPORTED: Indicates whether an RDA-server may re-establish Transport Connections that have been terminated by an RDA-client within an SQL-connection.

RESUME MODE: Indicates the RDA Suspend and Resume facility requested by the client, this can be one of:

- *none*, the Transport Connection will not be terminated and re-established,
- *client*, the RDA-client may terminate and re-establish the Transport Connection,
- *server*, the RDA-client may terminate the Transport Connection and the RDA-server will re-establish it.

CLIENT DISCONNECT INTERVAL: When Client RDA Suspend and Resume is selected, the maximum length of time (in seconds) that an RDA-client may delay between terminating a Transport Connection and re-establishing it.

CLIENT CONNECT INTERVAL: When Client RDA Suspend and Resume is selected, the minimum length of time (in seconds) that an RDA-client must delay between sending an encoding through a Transport Connection and terminating it with the intention of re-establishing it later.

NOTE 15 – An implementation need not support the RDA suspend and resume facility

5.6.2 RDA Encoding Facility

An Attribute is provided to permit an RDA-client to identify what encodings other than the Default Encoding are recognised by an RDA-server:

ENCODINGS SUPPORTED: The sum of the codes identifying encodings that are supported by the RDA-server on this SQL-connection as defined in clause 9.

NOTE 16 – Encodings have codes which are (small) powers of 2.

5.7 RDA Facilities for Transaction Co-ordination

The RDA Transaction Co-ordination facility is:

RDA Transaction Co-ordination: a mechanism for ensuring that Service User operations leave the SQL-environment in a consistent state.

The following subclause describes how features defined by this International Standard are used to provide the RDA Transaction Co-ordination facility.

5.7.1 RDA Transaction Co-ordination Facility

An RDA Operation RDAEndTran is provided which may be used to communicate transaction co-ordination requests.

Three types of RDA Transaction Co-ordination are supported:

- *encompassing*, transaction co-ordination is the responsibility of some external transaction management component. A field, MessageContext, is provided in the RDAMessage protocol element for the use of the external transaction management component to communicate transaction context information. RDAEndTran is not used except as required by the external transaction management component.
- *one-phase*, when a Service User requests that a transaction be committed, the request is communicated to each SQL-server participating in the transaction independently. This level of support may not preserve the transaction semantics of ISO/IEC 9075 (SQL) if the SQL-servers do not all commit the transaction. RDAEndTran is used to communicate COMMIT or ROLLBACK requests.
- *two-phase*, when a Service User requests that a transaction be committed, a multiphase commitment exchange is entered into with the SQL-servers participating in the transaction. This level of support preserves the transaction semantics of ISO/IEC 9075 (SQL) in multi-site transactions except possibly in the event of a breakdown during the final phase of the two-phase exchange in a server or in the communications. RDAEndTran is used to communicate PREPARE TO COMMIT, COMMIT or ROLLBACK requests.

Mechanisms to recover automatically from breakdowns in the final phase of a two-phase exchange are outside the scope of this International Standard.

Attributes are provided to permit an RDA-server to indicate what types of RDA Transaction Co-ordination it provides and a Service User to request an RDA Transaction Co-ordination Type:

ONE-PHASE CO-ORDINATION SUPPORTED: Indicates whether the server supports One-phase RDA Transaction Co-ordination.

TWO-PHASE CO-ORDINATION SUPPORTED: Indicates whether the server supports Two-phase RDA Transaction Co-ordination.

CO-ORDINATION REQUESTED: Indicates the type of RDA Transaction Co-ordination requested by the Service User.

5.8 RDA Facilities for Security

5.8.1 RDA Security Services

The requirements for security services are discussed in Annex F. These services are:

RDA Service User Authentication: corroboration of the identity of the Service User defined in 5.2.1.

RDA-client Authentication: corroboration of the identity of the RDA-client.

RDA-server Authentication: corroboration of the identity of the RDA-server.

RDA Outgoing Access Control: access control enforced by the RDA-client environment.

RDA Firewall Access Control: access control enforced by the RDA relay system

RDA Incoming Access Control: access control enforced by the RDA-server environment.

SQL Access Control: access control enforced by the SQL-server.

RDA Transfer Integrity: protection of communication against unauthorised modification.

RDA Transfer Confidentiality: protection of communication against unauthorised disclosure.

RDA Request Non-repudiation: protection against the RDA-client denying having requested a specific RDA Operation.

RDA Response Non-repudiation: protection against the RDA-server denying having responded to a specific request for an RDA Operation.

The following subclauses describe how facilities defined or identified by this International Standard may be used to provide the security services. Profiles of combinations of services to achieve defined levels of security are defined in Annex G.

NOTE 17 – In this International Standard RDA Outgoing Access Control, RDA Firewall Access Control and RDA Incoming Access Control are to a resource such as an SQL-server rather than to data. SQL Access Control is to data.

5.8.2 Use of Transport Provider security facilities

The Transport Provider may have facilities for providing RDA Transfer Integrity, RDA Transfer Confidentiality, RDA-client Authentication and RDA-server Authentication.

A Transport Mapping is defined in 10.2 that can be used to provide RDA Transfer Integrity, RDA Transfer Confidentiality, RDA-client Authentication and RDA-server Authentication.

In cases where the RDA Client represents a single Service User then RDA Client Authentication may be used to support RDA Service User Authentication as described in 5.8.3.

5.8.3 Use of Authentication in RDAConnect

The RDAConnect operation fields AuthenticationType and Authentication can be used to authenticate the name of the RDA Service User (defined in 5.2.1) as identified by the UserName field.

Four types of RDA Service User Authentication, identified by the AuthenticationType, are supported:

- *password*, the UserName is authenticated using a password carried in the Authentication field.
- *transfer*, the UserName is matched against the RDA-client identity authenticated by RDA-client Authentication (for example using a local equivalence table or mapping algorithm). This mechanism requires that the RDA-client represent a single Service User.

  NOTE 18 – The Service User may, however, use several User Names to represent different privileges and may be represented by several RDA-clients.
- *attributeCertificate*, the User Name is related to the identity of an RDA-client by an Attribute Certificate (as defined in ISO/IEC 9598-8 | X.509 1997) carried in the Authentication field. This mechanism requires that the RDA-client represent a single Service User.

  NOTE 19 – The Service User may, however, use several User Names to represent different privileges and may be represented by several RDA-clients. Also, if Service Users share common privileges, and do not need to be separately identified for other reasons (for example, accountability), then they can be treated as the same Service User for the purposes of authentication, sharing a common UserName authenticated through the same RDA-client. An amendment to this International Standard is in preparation that gives further details of how Attribute Certificates are used for the management of privileges.
- *other*, the Authentication field is used to carry authentication information to support an externally defined mechanism.

The AuthenticationType may also be set to *none* to indicate that no support is provided for RDA User Authentication.

An Attribute is provided to permit an RDA-client to request a particular type of RDA Service User Authentication:

AUTHENTICATION TYPE: a value defining the type of RDA Service User Authentication requested by the RDA-client environment.

5.8.4 Use of MessageAuthentication in RDAMessage

Each RDAMessage protocol element has a field MessageAuthentication that provides a means of communicating information required for RDA Request Non-repudiation and RDA Response Non-repudiation.

Three levels of RDA Request Non-repudiation and RDA Response Non-repudiation are supported:

- *none*, no support is provided
- *originatorSigned*, RDA Non-repudiation is supported by a timestamp and signature produced by the originator of the message
- *ttpSigned*, RDA Non-repudiation is supported by a timestamp and signature produced by the originator of the message, together with a timestamp and signature produced by a trusted third party (TTP).

Attributes (as defined in 5.4.1.1) are provided to permit an RDA-client and an RDA-server to negotiate the RDA Request Non-repudiation level and the RDA Response Non-repudiation level, and for a Service User to request a particular level of RDA Request Non-repudiation and RDA Response Non-repudiation.

REQUEST NON-REPUDIATION PROVIDED: a value defining the level of RDA Request Non-repudiation provided by the RDA-client environment.

REQUEST NON-REPUDIATION REQUIRED: a value defining the level of RDA Request Non-repudiation required by the RDA-server environment.

RESPONSE NON-REPUDIATION SUPPORTED: a value defining the level of RDA Response Non-repudiation supported by the RDA-client environment.

RESPONSE NON-REPUDIATION REQUIRED: a value defining the level of RDA Response Non-repudiation required by the RDA-client environment.

6 RDA Protocol

The RDA Protocol is defined in terms of:

- an exchange of protocol elements between an RDA-client environment and an RDA-server environment, as defined in 6.1,
- the encoding of the RDA protocol element, RDAMessage, as defined in 6.2,
- a set of operations that are invoked through the protocol, as identified in Table 3,
- the behaviour of the RDA-client and RDA-server when an RDA Operation is invoked, as defined in 6.3,
- the encoding of operation requests, operation results and component elements of these, as defined in clause 7,
- the encoding of the basic data types, as defined in clause 10 for each Transport Mapping.

6.1 The RDA Protocol Exchange

The RDA Protocol consists of:

- A protocol element sent by RDA-client Services from an RDA-client environment to an RDA-server environment carrying the encoding of a request for the execution of an RDA Operation by the RDA-server environment.
- A protocol element sent by RDA-server Services from an RDA-server environment to an RDA-client environment carrying a response resulting from the execution of an RDA Operation.

The RDA Protocol is a confirmed protocol. That is, the RDA-client expects to receive an appropriate protocol element in response to each protocol element it sends.

Protocol elements may be sent asynchronously. That is, RDA-client Services do not require that an RDA-client wait for a response to a protocol element before sending the next protocol element.

NOTE 20 – The logic of an application may require that an RDA Operation completes before the next RDA Operation is invoked.

NOTE 21 – An RDA-server environment may limit the number of RDA Operation invocations it can support concurrently.

Protocol elements may be concatenated. That is, a sender may collect several protocol elements and send them together.

NOTE 22 – The logic of an application may require that a response is received before the next operation request can be sent.

6.2 RDAMessage

6.2.1 RDAMessage protocol element

Function

Carry an RDA message.

Encoding

```
RDAMessage                    ::=   SEQUENCE
   {  MessageProtocol                   RDAInteger,
      MessageVersion                    RDAInteger,
      MessageEncoding                   RDAInteger,
      MessageLength                     RDAInteger,
      messageConnectionIdent            RDAInteger,
      MessageRequestIdent               RDAInteger,
      MessageType                       RDAInteger,
      MessageContext                    RDAOctetString,
      MessageData                       RDAOctetString,
      MessageAuthentication             RDAOctetString
   }
```

The RDAMessage protocol element is encoded according to the Base encoding defined in 9.1.

Protocol Parameters

MessageProtocol: a code identifying the RDA Protocol. Valid codes are defined in Table 1.

Table 1 – Codes used to identify the protocol

Protocol	Code
ISO/IEC 9579, Remote Database Access for SQL	959788857

NOTE 23 – The code 959788857 is the ASCII encoding for the four characters “9579”

MessageVersion: a code identifying the version of the RDA Protocol. Valid codes are defined in Table 2.

Table 2 – Codes used to identify the protocol version

Protocol	Version	Code
ISO/IEC 9579 Remote Database Access for SQL	ISO/IEC 9579:1998 Remote Database Access for SQL	3

MessageEncoding: a code identifying the encoding of the MessageData field. Valid codes are defined in clause 9.

MessageLength: the length in octets of the RDAMessage protocol element starting with the first octet following MessageLength.

NOTE 24 – Any octets following MessageAuthentication are ignored by this edition of this International Standard.

NOTE 25 – The protocol parameters MessageProtocol, MessageVersion, MessageEncoding and MessageLength will retain their semantics and encoding in future editions of this International Standard.

MessageRequestIdent: the Request Ident of this RDA message.

MessageType: a code identifying the type of this RDA message. Valid codes are defined in Table 3.

Table 3 – Codes used to identify an RDA message type

Message Type	Code
Request – RDAConnect	1001
Request – RDADisconnect	1002
Request – RDAEndTran	1003
Request – RDAClientAttribute	1004
Request – RDAStatementPrepare	1005
Request – RDAStatementDeallocate	1006
Request – RDAStatementExecute	1007
Request – RDAStatementExecDirect	1008
Request – RDAStatementFetchRows	1009
Request – RDAStatementCloseCursor	1010
Request – RDAStatementCancel	1011
Request – RDASetCursorName	1012
Request – RDAGetCursorName	1013
Request – RDAGetInfo	1014
Request – RDAGetTypeInfo	1015
Response	2001

MessageContext: an encoding of information required by the manager of an encompassing transaction, or the empty string if no encompassing transaction manager is in use.

MessageData: an encoding of an RDA request or response as defined in clause 7.

MessageAuthentication: an RDAOctetString whose Encoded Value is a MessageAuthentication encoding element as defined in clause 6.2.2, or an RDAOctetString encoding a string of length 0 if

communication is taking place without RDA Request non-repudiation and without RDA Response non-repudiation.

6.2.2 MessageAuthentication encoding element

Function

Encode a MessageAuthentication parameter

Encoding

```
MessageAuthentication          ::=   SEQUENCE
   {  MessageNonRepLevel                 RDAInteger,
      MessageResponseLevel               RDAInteger OPTIONAL,
      MessageTimestamp                   GeneralisedTime,
      OriginatorSignature                MessageAndTimeSig,
      OriginatorCertificate              CertificatePath,
      TtpSignature                       MessageAndTimeSig OPTIONAL,
      TtpCertificate                     CertificatePath OPTIONAL
   }

MessageAndTimeSig              ::=   SIGNATURE
   {  SEQUENCE
      {  MessageProtocol                 RDAInteger,
         MessageVersion                  RDAInteger,
         MessageEncoding                 RDAInteger,
         MessageLength                   RDAInteger,
         MessageRequestIdent             RDAInteger,
         MessageType                     RDAInteger,
         MessageContext                  RDAOctetString,
         MessageData                     RDAOctetString,
         MessageNonRepLevel              RDAInteger,
         MessageResponseLevel            RDAInteger OPTIONAL,
         MessageTimestamp                GeneralisedTime
      }
   }
```

Syntax and Encoding

The syntax of GeneralisedTime is defined in ISO/IEC 8824-1.

The syntax of CertificatePath and SIGNATURE are defined in ISO/IEC 9594-8 | ITU-T X.509.

The encoding of MessageAuthentication and its subfields is according to the Distinguished Encoding Rules (DER) of ISO/IEC 8825-1:1995.

NOTE 26 – This encoding is independent of the encoding chosen for the Transport Mapping

The Encoded Value of MessageAndTimeSig is the syntax of SIGNATURE applied to message fields encoded using the Default Encoding for RDA specific data types.

Encoding Parameters

MessageNonRepLevel: the non-repudiation level of this message. This is encoded as the value for a non-repudiation Attribute in Table 9.

MessageResponseLevel: if this message is a request for an RDA Operation then the value of the RESPONSE NON-REPUDIATION REQUIRED Attribute. This encoding parameter is not present if the message is a response.

Other encoding parameters may be not present depending on the value of MessageNonRepudiateLevel according to Table 4

Table 4 – Use of MessageAuthenticateParameters

	MessageNonRep Level		
Parameter	***none***	***originatorSigned***	***ttpSigned***
MessageTimestamp	not present	present	present
OriginatorSignature	not present	present	present
OriginatorCertificate	not present	present	present
TtpSignature	not present	not present	present
TtpCertificate	not present	not present	present

Those encoding elements which are defined to be present by MessageNonRepudiateLevel have a value defined below (which may be null):

MessageTimestamp: the Timestamp for this message

OriginatorSignature: a signature calculated by the originator of the message.

OriginatorCertificate: the certificate for the originator's public key, the syntax of CertificatePath is as defined in ISO/IEC 9594-8 | ITU-T X.509.

TtpSignature: a signature calculated by a Trusted Third Pary.

TtpCertificate: the certificate for the Trusted Third Party's public key, the syntax of CertificatePath is as defined in ISO/IEC 9594-8 | ITU-T X.509

6.3 Invocation of RDA Operations

The execution of RDA Operations takes place in four phases:

- Invocation of the request in the RDA-client environment
- Evaluation of the request in the RDA-server environment
- Invocation of the response in the RDA-server environment
- Evaluation of the response in the RDA-client environment

These four phases are defined in subsequent subclauses by means of a sequence of rules for each phase and a further sequence common to all four, the *Transport Fail Indication* rules. These are evaluated when a Transport Connection fails or the sending or receiving of an encoding is not successful.

6.3.1 Invocation of the Request in the RDA-client environment

Within an RDA-client environment an RDA Operation is invoked through RDA-client Services. This invocation may be by means of the Application Programming Interface defined in Annex B.

When an RDA Operation *RO* is invoked on an allocated SQL-connection *SC* between an RDA-client *RC* and an RDA-server *RS* through the RDA-client Services *RCS* of *RC*:

1) If there is no established Transport Connection associated with *RC* then the Transport Connect facility is invoked to establish a Transport Connection.
2) If the Transport Connection has not been established, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.
3) The parameters of *RO* are encoded into an RDAMessage protocol element *E* according to the specification in clause 9.
4) Valid values for MessageProtocol, MessageVersion, MessageEncoding are selected for *E* and encoded into *E*.
5) MessageLength is computed and encoded into *E*.
6) The MessageConnectionIdent is encoded into *E*.
7) A MessageRequestIdent is allocated for *RO* and encoded into *E*.
8) The MessageType is encoded into *E*.
9) If required, the MessageAuthentication field is computed and encoded into *E*, with MessageNonRepLevel set to the level being applied to this message, and MessageResponseLevel set to the level required in the response. Otherwise the MessageAuthentication field is set to the empty string.
10) Let *T* be the established Transport Connection associated with *RC* and let *RS* be the RDA-server with which *T* is associated.
11) The Transport Send facility is invoked to send *E* to *RS* through *T*.
12) If the sending of *E* was not successful, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.
13) Let *RM* be the value of RESUME MODE of *SC*.

14) If *RM* has value *none* then evaluation of these rules terminates.

15) Let *CCI* be the value of CLIENT CONNECT INTERVAL for *SC*.

16) Case:

a) If *RM* has value *client* and the CLIENT RESUME PERMITTED attribute of *SC* has value *true* then:

i) Let *CDI* be the value of CLIENT DISCONNECT INTERVAL for *SC*.

ii) If a period of time greater than *CCI* seconds has elapsed since Transport Send was last invoked for *T* then the Transport Disconnect facility may be invoked for *T*.

iii) If RDA Operations have been invoked through *T* for which no response has yet been received and *T* is no longer established then the Transport Connect facility is invoked before *CDI* seconds have elapsed since the Transport Disconnect facility was invoked for *T* and evaluation of these rules continues from 16)a)ii).

b) If *RM* has value *server* and the SERVER RESUME SUPPORTED attribute of *SC* has value *true* then the Transport Disconnect facility may be invoked for *T* when a period of time greater than *CCI* seconds has elapsed since *T* was established.

6.3.2 Evaluation of the Request in the RDA-server environment

On receipt of an encoding *E*:

1) Let the Transport Connection through which *E* is received be *T*.

2) If the Transport Receive facility indicates that the *T* has failed or *E* was not received correctly, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.

3) *E* is regarded as the encoding of an RDAMessage protocol element *M*.

4) If the value of MessageProtocol of *M* is not one of the codes in Table 1 then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates

5) If the value of MessageVersion is not one of the codes in Table 2 then the exception is raised: *RDA-specific condition – version not supported*

6) If the value of MessageEncoding is not supported then the exception is raised: *RDA-specific condition – encoding not supported*.

7) Let *MRT* be the value of MessageRequestType.

8) If *MRT* is not one of the codes in Table 3 designating a message type ‘Request’ then the exception is raised: *RDA-specific condition – invalid message type*.

9) Let *MRI* be the value of MessageRequestIdent.

10) If *MRI* is the same as the value of MessageRequestIdent of some other RDAMessage protocol element received by this RDA-server for which this RDA-server has not yet sent an RDAMessage response protocol element, then the exception is raised: *RDA-specific condition – duplicate request ident*.

11) If required, the MessageAuthentication field is checked. If the OriginatorSignature or TtpSignature verification fails then the exception is raised: *RDA-specific condition: authentication failure*. *M* may be archived for later protection against repudiation.

12) Let *MCI* be the value of MessageConnectionIdent.

13) The RDA Server Evaluation Rules for the RDA Operation identified by *MRT* are evaluated within the SQL-session identified by *MCI* regarding the value of MessageData as an encoding of that RDA Operation.

NOTE 27 – This evaluation may involve the execution of SQL statements by the SQL-server

6.3.3 Invocation of the Response in the RDA-server environment

Within an RDA-server environment an RDA Response Operation is invoked through RDA-server Services.

When the evaluation of an RDA Operation *RO* associated with and SQL-connection *SC* has completed:

1) Let *RS* and *RC* be the RDA-server and the RDA-client associated with *SC*.
2) The results of the evaluation are encoded into an RDAResponse encoding element as defined in 7.2 using the same encoding as *RO*, or the default encoding if the encoding of *RO* is not supported.
3) The RDAResponse encoding element is included as MessageData into an RDAMessage protocol element *E* according to the specification in 6.2 using the same values of MessageProtocol, MessageVersion, MessageEncoding, MessageConnectionIdent, MessageRequestIdent and MessageContext as *RO*.
4) Let *RL* be the MessageResponseLevel of *RO*.
5) If the RDA-server does not support the RDA Response Non-Repudiation Level *RL* then the exception is raised *RDA-specific condition: authentication failure*.
6) If *RL* is not none then the MessageAuthentication field is computed and encoded into E according to *RL*.
7) MessageNonRepLevel is set to *RL*.
8) If there is no established Transport Connection associated with *RC*, then:
   a) Let *RM* be the value of RESUME MODE of *SC*.
   b) Case:
      i) If *RM* has value *client* and the CLIENT RESUME PERMITTED attribute of *SC* has value *true* then:
         (1) Let *CDI* be the value of CLIENT DISCONNECT INTERVAL for *SC*
         (2) The evaluation of these rules is suspended until a Transport Connection becomes established between *RC* and *RS*. If a period of time exceeding *CDI* seconds elapses and no Transport Connection becomes established between *RC* and *RS* then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates
      ii) If *RM* has value *server* and the SERVER RESUME SUPPORTED attribute of *SC* has value *true* then the Transport Connect facility is invoked to establish a connection between *RS* and *RC*.
      iii) Otherwise, the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates
9) Let *T* be the established Transport Connection between *RS* and *RC*.
10) The encoding *E* is sent to *RC* using the Transport Send facility associated with *T* within *RS*.

11) If the sending of *E* was not successful, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.

6.3.4 Evaluation of the Response in the RDA-client environment

On receipt of an encoding *E*:

1) Let the Transport Connection through which *E* is received be *T*.
2) If the Transport Receive facility indicates that the *T* has failed or *E* was not received correctly, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.
3) *E* is regarded as the encoding of an RDAMessage protocol element *M*.
4) If the value of MessageProtocol of *M* is not one of the codes in Table 1 then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates
5) If any of the following conditions hold:
   a) the value of MessageProtocol is not one of the codes in Table 1,
   b) the value of MessageVersion is not one of the codes in Table 2,
   c) the value of MessageEncoding is not valid as defined in 6.2,
   d) the value of MessageConnectionIdent does not identify an SQL-connection,
   e) the value of MessageRequestIdent does not designate the message type ‘Response’,

   then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.
6) Let *MRI* be the value of MessageRequestIdent.
7) If *MRI* is not equal to the value of the RequestIdent of an RDARequest protocol that has been sent by the RDA-client for which no response has previously been received, then the rules for Transport Fail Indication are evaluated and evaluation of these rules terminates.
8) If required, the MessageAuthentication field is checked. If the OriginatorSignature or TtpSignature verification fails then the exception is raised: *RDA-specific condition: authentication failure. M* may be archived for later protection against repudiation.
9) The RDA-client Operation Rules for the RDAResponse encoding element in 7.2 are evaluated, regarding the value of MessageData as an encoding of RDAResponse.

6.3.5 Transport Fail Indication

In the event of a Transport Connection *T* experiencing a failure or the sending or receiving of an encoding through *T* is not being successful within an RDA-client *RT* or an RDA-server *RT*:

1) All encodings in the process of being sent by *RT* through *T* are discarded.
2) The Transport Fail facility is invoked for *T*.
3) All encodings in the process of being received by *RT* through *T* are discarded.
4) All encodings subsequently sent by *RT* through *T* are discarded.
5) If *RT* is an RDA-client then each RDAMessage sent through *T* which is discarded, or for which no response has yet been received, is deemed to have received in response the exception *RDA-specific condition – transport failure*.

6) If *RT* is an RDA-server then RDADisconnect is deemed to have been received.

7 RDA Operations

7.1 RDA request operations

7.1.1 RDAConnect Operation

Function

Establish an SQL-connection

Encoding

```
RDAConnect                    ::=   SEQUENCE
   {  destinationServerName           RDACharString,
      userName                        RDACharString,
      authenticationType              RDAInteger,
      authentication                  RDAOctetString
   }
```

Encoding Parameters

DestinationServerName: the SQL-server name of the SQL-server with which an SQL-connection is to be established.

UserName: the SQL-user name of the SQL-session that this SQL-connection will support.

AuthenticationType: the value of the AUTHENTICATION TYPE Connection Attribute for the SQL-connection.

Authentication: a value used to authenticate the SQL-user name. The interpretation of this value depends upon the value of AuthenticationType as follows:

- *none*, the value of Authentication is ignored,
- *password*, Authentication carries a password encoded as an RDACharString,
- *transfer*, the transport mapping is to a Transport Provider that authenticates the RDA-client. The RDA-server can validate that the RDA-client can rightfully use UserName (for example using a local equivalence table or mapping algorithm). The value of Authentication is ignored
- *attributeCertificate*, Authentication carries an Attribute Certificate as defined in ISO/IEC 9594-8 | ITU-T X.509, with the subject field set to the RDA-client name (that is the identity of the RDA-client authenticated by the transport mapping) and the attribute field set to UserName. The Transport Mapping is to a Transport Provider that authenticates the RDA-client.

  NOTE 26 – This field is encoded according to the Distinguished Encoding Rules (DER) of ISO/IEC 8825-1.

– *other*, Authentication carries a field of type SEQUENCE { OBJECT IDENTIFIER, AuthenticationInformation} where the syntax and semantics of AuthenticationInformation are as identified by the OBJECT IDENTIFIER.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation and *RC* the RDA-client requesting the operation.
2) If a connection handle is already associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence.*
3) A Connection Handle which identifies an allocated SQL-connection that does not have an associated established SQL-connection and is not associated with an RDA-server is found.

   NOTE 29 – It is implementation dependent whether the General Rules of SQLAllocEnv, SQLAllocConnect, SQLAllocHandle are evaluated or whether the Connection Handle is found by other means. This permits re-use of environment handles and connection handles.
4) If no such Connection Handle can be found, the exception is raised *RDA-specific condition – no connection handle available.*
5) Let *C* be the Connection Handle so found, *SN* be the value of DestinationServerName, *UN* be the value of UserName, *AT* the value of AuthenticationType and *AU* be the value of Authentication.
6) *C* is associated with *RS*.
7) If *AT* is not acceptable according to the security policy, then the exception is raised: *RDA-specific condition – authentication failure.*
8) Case:
   a) If AT is transfer:
      i) If RC has not been authenticated by the Transport Provider then the exception is raised: RDA-specific condition: authentication failure.
      ii) If RC cannot rightfully use UN, then the exception is raised: RDA-specific condition: authentication failure.
   b) If AT is password and the security policy indicates that password checks are to be carried out by the RDA-server rather than the SQL-server and the password check fails, then the exception is raised: RDA-specific condition: authentication failure.
   c) If AT is attributeCertificate:
      i) If the Attribute Certificate AU fails to be validated, then the exception is raised: RDA-specific condition: authentication failure.
      ii) If RC has not been authenticated by the Transport Provider then the exception is raised: RDA-specific condition: authentication failure.
      iii) If the subject of Attribute Certificate AU is not RC then the exception is raised: RDA-specific condition: authentication failure.
      iv) If the attribute of Attribute Certificate AU is not UN then the exception is raised: RDA-specific condition: authentication failure.

9) The General Rules of SQLConnect are evaluated with ConnectionHandle set to *C*, SQL-server name set to *SN*, UserName set to *UN* and Authentication set to *AU*.

7.1.2 RDADisconnect Operation

Function

Terminate an established SQL-connection

Encoding

```
RDADisconnect                    ::=   SEQUENCE
   {
   }
```

Encoding Parameters

none

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *C* be the Connection Handle associated with *RS*.
4) The association between *RS* and *C* is discontinued.
5) The General Rules of SQLDisconnect are evaluated with ConnectionHandle set to *C*.

NOTE 30 – It is implementation dependent whether the General Rules of any of SQLFreeConnect, SQLFreeEnv and SQLFreeHandle are evaluated.

7.1.3 RDAEndTran Operation

Function

Terminate an SQL-transaction or prepare an SQL-transaction for termination

Encoding

```
RDAEndTran                    ::=   SEQUENCE
   {  completionType                     RDAInteger
   }
```

Encoding Parameters

CompletionType: the type of completion requested by the RDA-client environment. The values of CompletionType are those defined in Table 14 of ISO/IEC 9075-3 (SQL/CLI) together with the entry in Table 5.

Table 5 – Extension to Table 14 of ISO/IEC 9075-3

Termination Type	Code
PREPARE TO COMMIT	3

NOTE 31 – The way in which an SQL-client maps SQLEndTran onto RDAEndTran is defined in C.2.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *C* be the Connection Handle associated with *RS* and let *CT* be the value of CompletionType.
4) Case:
   a) If *CT* is COMMIT or ROLLBACK the General Rules of SQLEndTran are evaluated with HandleType set to the code for CONNECTION HANDLE, Handle set to *C* and CompletionType set to *CT*.
   b) If *CT* is PREPARE TO COMMIT

      Case:

      i) If the Connection Attribute TWO-PHASE CO-ORDINATION SUPPORTED has value *false*, then the exception is raised: *RDA-specific condition – feature not supported – multiple server transactions*.

ii) If a subsequent evaluation of the General Rules of SQLEndTran with HandleType set to the code for CONNECTION HANDLE, Handle set to *C* and CompletionType set to COMMIT would raise an exception, then the exception is raised *RDA-specific condition – transaction cannot commit*.

NOTE 32 – Determining this case may involve the server in attempting to prepare the transaction for commitment.

NOTE 33 – If a subsequent evaluation of the General Rules of SQLEndTran with HandleType set to the code for CONNECTION HANDLE, Handle set to *C* and CompletionType set to COMMIT would not raise an exception, then evaluation of these rules terminates.

c) Otherwise, the exception is raised: *RDA-specific condition – invalid transaction operation code*.

7.1.4 RDAClientAttribute Operation

Function

Synchronise RDA-server environment Attributes with RDA-client environment Attributes.

Encoding

```
RDAClientAttribute              ::=   SEQUENCE
   {  clientAttributes                    RDAAttributeList
   }
```

Encoding Parameters

ClientAttributes: a list of Attributes and their values.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *CH* be the Connection Handle associated with *RS* and let *EH* the Environment Handle with which *CH* is associated.
4) For each Attribute *A* in ClientAttributes:
   a) Let *AT* be the value of AttributeType of *A*, *AC* be the value of AttributeCode, *AV* the value of AttributeValue and *SI* the value of StatementIdent.
   b) If *AC* is not the code for a permitted Attribute of type *AT* as defined in the AttributeCode Encoding parameter definition of RDAAttribute in 7.3 then the exception is raised: *RDA-specific condition – attribute not permitted*.
   c) Case:
      i) If *AT* is the code for Environment Attribute, then the General Rules of SetEnvAttr are evaluated with EnvironmentHandle set to *EH*, Attribute set to *AC* and Value set to *AV*.
      ii) If *AT* is the code for Connection Attribute, then then the General Rules of SetConnectAttr are evaluated with ConnectionHandle set to *CH*, Attribute set to *AC* and Value set to *AV*.
      iii) If *AT* is the code for Statement Attribute, then:
         (1) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
         (2) Let the Statement Handle associated with *SI* be *SH*.
         (3) the General Rules of SetStmtAttr are evaluated with StatementHandle set to *SH*, Attribute set to *AC* and Value set to *AV*.

iv) Otherwise, the exception is raised: *RDA-specific condition – invalid attribute type*

7.1.5 RDAStatementPrepare Operation

Function

Prepare an SQL statement.

Encoding

```
RDAStatementPrepare           ::=   SEQUENCE
   {  statementIdent                     RDAInteger,
      statementText                      RDACharString
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify this statement to subsequent RDAStatement operations. StatementIdent may not have the value NO STATEMENT as defined in Table 10.

StatementText: the text of the statement to be prepared.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *CH* be the Connection Handle associated with *RS*, let *SI* be the value of StatementIdent and let *ST* the value of StatementText.
4) If there is no Statement Handle associated with *SI* then:
   a) The General Rules of SQLAllocStmt are evaluated with ConnectionHandle set to *CH*.
   b) The resulting Statement Handle is associated with *SI*.
5) Let *SH* be the Statement Handle associated with *SI*.
6) The General Rules of SQLPrepare are evaluated with StatementHandle set to *SH* and StatementText set to *ST*.

7.1.6 RDAStatementDeallocate Operation

Function

Deallocate an SQL-statement.

Encoding

```
RDAStatementDeallocate          ::=   SEQUENCE
   {  statementIdent                     RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the statement to be deallocated.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* the Statement Handle associated with *SI*.
6) The General Rules of SQLFreeStmt are evaluated with StatementHandle set to *SH* and Option set to the code for FREE HANDLE.

7.1.7 RDAStatementExecute Operation

Function

Execute a prepared statement.

Encoding

```
RDAStatementExecute              ::=   SEQUENCE
   {  statementIdent                         RDAInteger,
      parameterDescriptor                    RDAItemDescriptorList,
      parameterData                          RDARowList
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the statement to be executed.

ParameterDescriptor: fields from the Application Parameter Descriptor for this statement. This describes the values of each RDARow in ParameterData.

ParameterData: sets of data from the Application Parameter Descriptor for this statement.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* be the Statement Handle associated with *SI*.
6) For each RDARow item *ROW* in ParameterData:
   a) Let *ND* be the number of RDAItemDescriptors in ParameterDescriptor and let *NV* be the number of RDAValues in *ROW*.
   b) Let *DH* be the descriptor handle of the current Application Parameter Descriptor of *SH*.
   c) If *ND* and *NV* are both 0, then the General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to 0, FieldIdentifier set to COUNT and Value set to 0.
   d) If *ND* is greater than 0, then:
      i) If *ND* does not equal *NV* then the exception is raised: *RDA-specific condition – number of values does not match number of item descriptors*.

ii) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to 0, FieldIdentifier set to COUNT and Value set to ND.

iii) For each RDAItemDescriptor, *ID*, in ParameterDescriptor:

(1) Let *RN* be the ordinal position of *ID* in ParameterDescriptor.

(2) For each RDADescriptorEntry *DE* in *ID*:

(a) Let *DC* be the value of DescriptorCode of *DE* and *DV* be the DescriptorValue of *DE*.

(b) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to *DC* and Value set to *DE*.

e) If *ND* is 0 and *NV* is greater than 0, then

i) Let *NAPD* be the setting of Value resulting from the evaluation of the General Rules for SQLGetDescField with DescriptorHandle set to *DH*, RecordNumber set to 0 and FieldIdentifier set to COUNT.

ii) If *NV* is not equal to *NAPD* then the exception is raised: *RDA-specific condition – number of values does not match number of item descriptors*.

f) If *NV* is greater than 0, then:

i) For each RDAValue, *IV*, in ParameterData:

(1) Let *RN* be the ordinal position of *IV* in ParameterData.

(2) Let *T* be the encoding choice of *IV*, *L* be the encoding length of *IV*, *CL* be the character length of *IV* and *V* be the encoding value of *IV*.

(3) Case:

(a) If *T* is NullValue then the General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to INDICATOR_POINTER and Value set to -1.

(b) If *T* is CharacterValue or BitValue then:

(i) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to INDICATOR_POINTER and Value set to 0.

(ii) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to OCTET_LENGTH and Value set to *L*.

(iii) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to LENGTH and Value set to *CL*.

(iv) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to OCTET_LENGTH_POINTER and Value set to *L*.

(v) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to LENGTH_POINTER and Value set to *CL*.

(vi) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to DATA_POINTER and Value set to *V*.

(c) If *T* is IntegerValue or RealValue then:

(i) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to INDICATOR_POINTER and Value set to 0.

(ii) The General Rules of SQLSetDescField are evaluated with DescriptorHandle set to *DH*, RecordNumber set to *RN*, FieldIdentifier set to DATA_POINTER and Value set to *V*.

g) The General Rules of SQLExecute are evaluated with StatementHandle set to *SH*.

7.1.8 RDAStatementExecDirect Operation

Function

Execute a statement directly.

Encoding

```
RDAStatementExecDirect        ::=   SEQUENCE
   {  statementIdent                     RDAInteger,
      statementText                      RDACharString,
      parameterDescriptor                RDAItemDescriptorList,
      parameterData                      RDARowList
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify this statement to subsequent RDAStatement operations. StatementIdent may not have the value NO STATEMENT as defined in Table 10.

StatementText: the text of the statement to be executed.

ParameterDescriptor: fields from the Application Parameter Descriptor for this statement. This describes the values in each RDARow of ParameterData.

ParameterData: sets of data from the Application Parameter Descriptor for this statement.

RDA-server Evaluation Rules

1) The RDA-server Evaluation Rules for RDAStatementPrepare in 7.1.5 are evaluated.
2) The RDA-server Evaluation Rules for RDAStatementExecute in 7.1.7 are evaluated starting with the rule numbered “6)”.

7.1.9 RDAStatementFetchRows Operation

Function

Retrieve rows from a cursor.

Encoding

```
RDAStatementFetchRows          ::=   SEQUENCE
   {  statementIdent                     RDAInteger,
      fetchOrientation                   RDAInteger,
      fetchOffset                        RDAInteger,
      fetchCount                         RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the statement from whose cursor rows are to be retrieved.

FetchOrientation: a code indicating how the cursor is to be positioned prior to rows being retrieved. The value of this parameter is one of the codes in Table 21 of ISO/IEC 9075-3 (SQL/CLI).

FetchOffset: a number used to qualify FetchOrientation.

FetchCount: the number of rows requested.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* the Statement Handle associated with *SI*.
6) Let *FC* be the value of FetchCount.
7) If *FC* is less than 1 then the exception is raised: *RDA-specific condition – invalid fetch count*.
8) Let *FOR* be the value of FetchOrientation and *FOF* be the value of FetchOffset.
9) The General Rules of SQLFetchScroll are evaluated with StatementHandle set to *SH*, FetchOrientation set to *FOR* and FetchOffset set to *FOF*.
10) Let *FOR2* have the value NEXT unless *FOR* has the value LAST or the value PRIOR.

11) If *FC* is greater than 1, then the General Rules of SQLFetchScroll are evaluated (*FC* - 1) times with StatementHandle set to *SH*, FetchOrientation set to *FOR2* and FetchOffset set to 1

7.1.10 RDAStatementCloseCursor Operation

Function

Close a Cursor.

Encoding

```
RDAStatementCloseCursor          ::=   SEQUENCE
   {  statementIdent                        RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the statement whose cursor is to be closed.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* be the Statement Handle associated with *SI*.
6) The General Rules of SQLCloseCursor are evaluated with StatementHandle set to *SH*.

7.1.11 RDAStatementCancel Operation

Function

Attempt to cancel the execution of an SQL/CLI routine.

Encoding

```
RDAStatementCancel              ::=   SEQUENCE
   {  statementIdent                     RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the statement upon which the SQL/CLI routine whose execution is to be cancelled is operating.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* be the Statement Handle associated with *SI*.
6) The General Rules of SQLCancel are evaluated with StatementHandle set to *SH*.

NOTE 34 – The RDAStatementCancel operation may be invoked from a different execution thread to the operation that is being cancelled.

7.1.12 RDASetCursorName Operation

Function

Set a cursor name.

Encoding

```
RDASetCursorName                ::=   SEQUENCE
   {  statementIdent                     RDAInteger,
      cursorName                         RDACharString
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the SQL-statement whose cursor name is to be returned.

CursorName: the name to which the cursor name will be set.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* the Statement Handle associated with *SI* and *CN* the value of CursorName.
6) The General Rules of SQLSetCursorName are evaluated with StatementHandle set to *SH* and CursorName set to *CN*.

7.1.13 RDAGetCursorName Operation

Function

Get a cursor name.

Encoding

```
RDAGetCursorName                ::=   SEQUENCE
   {  statementIdent                     RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the SQL-statement whose cursor name is to be returned.

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* the Statement Handle associated with *SI*.
6) The General Rules of SQLGetCursorName are evaluated with StatementHandle set to *SH*.

7.1.14 RDAGetInfo Operation

Function

Get information about the SQL implementation.

Encoding

```
RDAGetInfo                    ::=   SEQUENCE
   {  InfoType                            RDAInteger
   }
```

Encoding Parameters

InfoType: a code specifying the information requested as defined in ISO/IEC 9075-3 (SQL/CLI).

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *IT* the value of InfoType.
4) The General Rules of SQLGetInfo are evaluated with InfoType set to *IT*.

7.1.15 RDAGetTypeInfo Operation

Function

Get information about supported data types.

Encoding

```
RDAGetTypeInfo                    ::=   SEQUENCE
   {  statementIdent                          RDAInteger,
      dataType                                RDAInteger
   }
```

Encoding Parameters

StatementIdent: a Statement Ident used to identify the SQL-statement that will return the requested information.

DataType: a code specifying the information requested as defined in ISO/IEC 9075-3 (SQL/CLI).

RDA-server Evaluation Rules

1) Let *RS* be the RDA-server evaluating the operation.
2) If there is no Connection Handle associated with *RS*, then the exception is raised: *RDA-specific condition – invalid operation sequence*.
3) Let *SI* be the value of StatementIdent.
4) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.
5) Let *SH* the Statement Handle associated with *SI*, and *DT* the value of DataType.
6) The General Rules of SQLGetTypeInfo are evaluated with StatementHandle set to *SH* and DataType set to *DT*.

7.2 RDA response encoding element

Function

Encode the results of an RDA Operation.

Encoding

```
RDAResponse                    ::=   SEQUENCE
   {  diagnostics                        RDADiagnostic,
      serverAttributes                   RDAAttributeList,
      parameterDescriptor                RDAItemDescriptorList,
      rowDescriptor                      RDAItemDescriptorList,
      rows                               RDARowList
   }
```

Encoding Parameters

Diagnostics: a Diagnostic Area generated as a result of executing the sequence of RDA-server Rules appropriate for the RDARequest for which this RDAResponse protocol element is a response.

ServerAttributes: a list of Attribute values known to the RDA-server environment. This list includes as a minimum:

1) If this is the first response communicated through this SQL-connection, then those attributes whose value has been set to differ from the default values as defined in ISO/IEC 9075-3 (SQL/CLI) other than at the request of the RDA-client.
2) Otherwise, those attributes that have been set by the RDA-server environment (other than at the request of the RDA-client) since the last response from this RDA-server to the RDA-client through this SQL-connection.

ParameterDescriptor: the Implementation Parameter Descriptor resulting from the successful preparation of a statement for execution.

RowDescriptor: the Implementation Row Descriptor resulting from the successful preparation of a statement for execution or the Application Row Descriptor resulting from the successful retrieval of rows.

Rows: data from the Application Row Descriptor resulting from the successful retrieval of rows.

RDA-client Evaluation Rules

1) Let *CI* be the Connection Ident of the operation.
2) Let *CH* be the Connection Handle in the RDA-client environment that is associated with *CI*.
3) Let *EH* be the Environment Handle that is associated with *CH*.

4) The RDA-client environment copy of the Diagnostic area associated with *CH* is emptied.

5) For each Attribute *A* in ServerAttributes:

   a) Let *AT* be the value of AttributeType of *A*, *AC* be the value of AttributeCode of *A*, *AV* be the value of AttributeValue of *A* and *SI* the value of StatementIdent of *A*.

   b) Case:

      i) If *AT* is the code for Environment Attribute, then the RDA-client Environment copy of the Environment Attribute *AC* for the SQL-environment is set to *AV*.

      ii) If *AT* is the code for Connection Attribute, then the Connection Attribute *AC* for ConnectionHandle *CH* is set to *AV*.

      iii) If *AT* is the code for Statement Attribute, then:

         (1) If there is no Statement Handle associated with *SI* then the exception is raised: *RDA-specific condition – invalid operation sequence*.

         (2) Let the Statement Handle associated with *SI* be *SH*.

         (3) the General Rules of SetStmtAttr are evaluated with StatementHandle set to *SH*, Attribute set to *AC* and Value set to *AV*.

      iv) Otherwise, the exception is raised: *RDA-specific condition – invalid attribute type*.

6) For each of the parameters DynamicFunction, DynamicFunctionCode, More, ReturnCode and RowCount in Diagnostics, *P*, the corresponding field in the RDA-client environment copy of the Diagnostic Area is set to the encoded value of *P*.

7) Let the encoded count of StatusRecord be *N*.

   a) the RDA-client environment copy of the Diagnostic Area is set to have *N* status records.

   b) The NUMBER field of the RDA-client environment copy of the Diagnostic Area is set to *N*.

8) For each RDADiagnosticStatus, *S*, in StatusRecord:

   a) Let *RN* be the ordinal position of *S* in StatusRecord.

   b) Let *RD* be the *RN*th status record of the RDA-client environment copy of the Diagnostic Area.

   c) For each parameter *PS* in *S*, the corresponding field of *RD* is set to the encoded value of *PS*.

   d) The MESSAGE_LENGTH field of *RD* is set to the encoded character length of MessageText of *S*.

   e) The MESSAGE_OCTET_LENGTH field of *RD* is set to the encoded length of MessageText of *S*.

9) Let *NP* be the encoded count of ParameterDescriptorList.

10) If *NP* is greater than 0, then

   a) If *RQ* did not specify a Statement Ident, then the exception is raised*: RDA-specific condition – unexpected parameter descriptor*.

   b) Let *SH* be the statement handle associated with the Statement Ident specified in *RQ*.

   c) Let *IPD* be the Implementation Parameter Descriptor of *SH*.

   d) *IPD* is emptied.

e) *IPD* is set to have *NP* item descriptors.

f) The value of COUNT for *IPD* is set to *NP*

g) For each RDAItemDescriptor *PI* of ParameterDescriptorList

i) Let *NPI* be the ordinal position of *PI* in ParameterDescriptorList

ii) For each RDADescriptorEntry, *PE*, of *PI* the corresponding field the *NPI*th item descriptor of *IPD* is set to the encoded value of the DescriptorValue of *PE*.

h) The field UNNAMED of *IPD* is set to *true* if the NAME field of *IPD* has zero length.

NOTE 35 – When *NP* is equal to zero the previously sent *IPD* (if any) is retained.

11) Let *NR* be the encoded count of RowDescriptorList.

12) If *NR* is greater than 0, then

a) If *RQ* did not specify a Statement Ident, then the exception is raised*: RDA-specific condition – unexpected row descriptor*.

b) Let *SH* be the statement handle associated with the Statement Ident specified in *RQ*.

c) Let *IRD* be the Implementation Row Descriptor of *SH*.

d) *IRD* is emptied.

e) *IRD* is set to have *NR* item descriptors.

f) The value of COUNT for *IRD* is set to *NR*.

g) For each RDAItemDescriptor *RI* of RowDescriptorList

i) Let *NRI* be the ordinal position of *RI* in RowDescriptorList

ii) For each RDADescriptorEntry, *RE*, of *RI* the corresponding field the *NRI*th item descriptor of *IRD* is set to the encoded value of the DescriptorValue of *RE*.

h) The field UNNAMED of *IRD* is set to *true* if the NAME field of *IRD* has zero length.

NOTE 36 – When *NR* is equal to zero the previously sent *IRD* (if any) is retained.

13) Let *N* be the encoded count of Rows.

14) If *N* is greater than 0 then

a) If *RQ* did not specify a Statement Ident, then the exception is raised*: RDA-specific condition – unexpected rows*.

b) Let *SH* be the statement handle associated with the Statement Ident specified in *RQ*.

c) Let *IRD* be the Implementation Row Descriptor of *SH*.

d) Let *R* be the first RDARow in Rows.

e) If the encoded count of *R* differs from the value of COUNT for IRD then the exception is raised: *RDA-specific condition – mismatch between descriptor and row.*

f) For each RDAValue, *C* in Values of *R*

i) Let *N* be the ordinal position of *C* in *R*.

ii) Let *D* be the *N*th item descriptor of *IRD*.

iii) Case:

(1) If RDAValue is NullValue, then INDICATOR_POINTER of *D* is set to point to the value -1.

(2) Otherwise,

(a) INDICATOR_POINTER of *D* is set to point to the value 0

(b) DATA_POINTER of *D* is set to point to a value equal to the encoded value of *C*

(c) If the encoded type of *C* is RDAOctetString, then OCTET_LENGTH of *D* is set to point to a value equal to the encoded length of *C*

(d) If the encoded type of *C* is RDACharString, the LENGTH of *D* is set to point to a value equal to the encoded character length of *C*

7.3 Encoding components

7.3.1 RDAAttribute encoding element

Function

Encode an Attribute value

Encoding

```
RDAAttribute                    ::=   SEQUENCE
   {  attributeType                      RDAInteger,
      attributeCode                      RDAInteger,
      attributeValue                     RDAInteger,
      statementIdent                     RDAInteger
   }
```

Encoding Parameters

AttributeType: used to identify whether the attribute is an Environment Attribute, a Connection Attribute or a Statement Attribute. The value of AttributeType is defined in Table 6.

Table 6 – Codes used for attribute types

Attribute Type	Code
ENVIRONMENT ATTRIBUTE	1
CONNECTION ATTRIBUTE	2
STATEMENT ATTRIBUTE	3

AttributeCode: used to designate a particular attribute of the Environment, Connection or Statement as indicated by AttributeType. The values of AttributeCode are those defined in Tables 15 of ISO/IEC 9075-3 (SQL/CLI), Table 16 of ISO/IEC 9075-3 (SQL/CLI) augmented by the entry in Table 7 and Table17 of ISO/IEC 9075-3 (SQL/CLI) with the exception that AttributeCode may not be the code for one of the prohibited attributes listed in Table 8.

Table 7 – Codes used for RDA defined Connection Attributes

Attribute	Code	May be set
ENCODINGS SUPPORTED	-111	No
ONE-PHASE CO-ORDINATION SUPPORTED	-112	No
TWO-PHASE CO-ORDINATION SUPPORTED	-113	No
CO-ORDINATION REQUESTED	-114	Yes

CLIENT RESUME PERMITTED	-115	No
SERVER RESUME SUPPORTED	-116	No
RESUME MODE	-117	Yes
CLIENT DISCONNECT INTERVAL	-118	Yes
CLIENT CONNECT INTERVAL	-119	No
AUTHENTICATION TYPE	-120	Yes

Table 8 – Prohibited attributes

Attribute Type	**Attribute**
Statement attribute	APD HANDLE
Statement attribute	ARD HANDLE
Statement attribute	IPD HANDLE
Statement attribute	IRD HANDLE

AttributeValue: the current value of the attribute designated by AttributeType and AttributeCode. The values of AttributeValue are those defined in Table 19 of ISO/IEC 9075-3 (SQL/CLI) augmented by the entry in Table 9.

Table 9 – Extension to Table 19 of ISO/IEC 9075-3

Attribute	**Data type**	**Values**
ENCODINGS SUPPORTED	INTEGER	*see* 5.6.2
ONE PHASE CO-ORDINATION SUPPORTED	INTEGER	0 (*false*); 1 (*true*)
TWO PHASE CO-ORDINATION SUPPORTED	INTEGER	0 (*false*); 1 (*true*)
CO-ORDINATION REQUESTED	INTEGER	0 (*encompassing*); 1 (*one-phase*); 2 (*two-phase*)
RESUME MODE	INTEGER	0 (*none*); 1 (*client*); 2 (*server*)
CLIENT RESUME PERMITTED	INTEGER	0 (*false*); 1 (*true*)
SERVER RESUME PERMITTED	INTEGER	0 (*false*); 1 (*true*)
CLIENT DISCONNECT INTERVAL	INTEGER	
CLIENT CONNECT INTERVAL	INTEGER	
CURSOR NAME	CHARACTER(255)	
AUTHENTICATION TYPE	INTEGER	0 (*none*); 1 (*password*); 2 (*transfer*);

		3 (*attributeCertificate*); 4 (*other*)
REQUEST NON-REPUDIATION PROVIDED	INTEGER	0 (*none*); 1 (*originatorSigned*); 2 (*ttpSigned*)
REQUEST NON-REPUDIATION REQUIRED	INTEGER	0 (*none*); 1 (*originatorSigned*); 2 (*ttpSigned*)
RESPONSE NON-REPUDIATION SUPPORTED	INTEGER	0 (*none*); 1 (*originatorSigned*); 2 (*ttpSigned*)
RESPONSE NON-REPUDIATION REQUIRED	INTEGER	0 (*none*); 1 (*originatorSigned*); 2 (*ttpSigned*)

StatementIdent: If the Attribute Type has the value STATEMENT ATTRIBUTE then the value of StatementIdent identifies the SQL-statement that this Attribute relates to. Otherwise, this parameter has the value NO STATEMENT as defined in Table 10.

Table 10 – Values of Statement Ident

Statement Ident	**Value**
NO STATEMENT	0

7.3.2 RDADiagnostic and RDADiagnosticStatus encoding elements

Function

Encode the Diagnostic Area.

Encoding

```
RDADiagnostic                ::=   SEQUENCE
   {  dynamicFunction                   RDACharString,
      dynamicFunctionCode               RDAInteger,
      more                              RDAInteger,
      returnCode                        RDAInteger,
      rowCount                          RDAInteger,
      statusRecords                     RDADiagnosticRecordList
   }

RDADiagnosticRecord          ::=   SEQUENCE
   {  diagnosticRecords                 RDADiagnosticStatusList
   }

RDADiagnosticStatus          ::=   SEQUENCE
   {  diagnosticCode                    RDAInteger,
      diagnosticValue                   RDAValue
   }
```

Encoding Parameters

StatusRecords: a list of diagnostic status records.

Other operation parameters in RDADiagnostic correspond to the fields with the same name in the CLI diagnostics area are defined in ISO/IEC 9075-3 (SQL/CLI). The matching of names for this purpose ignores capitalisation, and underscores.

DiagnosticRecords: a list of diagnostic status fields.

DiagnosticCode: a code used to designate a particular field of a diagnostic status record. The values of DescriptorCode are those defined in Table 12 of ISO/IEC 9075-3 (SQL/CLI) excluding MESSAGE_LENGTH and MESSAGE_OCTET_LENGTH.

DiagnosticValue: the value of the descriptor field identified by DiagnosticCode with type defined in ISO/IEC 9075-3 (SQL/CLI).

NOTE 37 – The value of MESSAGE_LENGTH and MESSAGE_OCTET_LENGTH can be deduced from the encoding.

7.3.3 RDAItemDescriptor encoding element

Function

Encode an Item Descriptor

Encoding

```
RDAItemDescriptor              ::=   SEQUENCE
   {  descriptorEntries                  RDADescriptorEntryList
   }

RDADescriptorEntry             ::=   SEQUENCE
   {  descriptorCode                     RDAInteger,
      descriptorValue                    RDAValue
   }
```

Encoding Parameters

DescriptorEntries: a list of descriptor entries.

DescriptorCode: a code used to designate a particular field of a descriptor. The values of DescriptorCode are those defined in Table 20 of ISO/IEC 9075-3 (SQL/CLI).

The entries LENGTH, NULLABLE and TYPE shall always be present.

If an RDADescriptorEntry with DescriptorCode designating TYPE is present with DescriptorValue *V* in an RDAItemDescriptor *D*, then the RDADescriptorEntries identified in Table 11 shall also be present in *D*, according to *V*.

Table 11 – RDADescriptorEntries required for SQL Data Types

SQL Data Type identified by *V*	**RDADescriptorEntries required**
CHARACTER (*L*)	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
CHARACTERVARYING (*L*)	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
BIT (*L*)	
BITVARYING (*L*)	
SMALLINT	
INTEGER	
DECIMAL (*P*, *S*)	PRECISION SCALE

NUMERIC (*P*, *S*)	PRECISION SCALE
REAL	
DOUBLEPRECISION	
FLOAT (*P*)	PRECISION
<datetime type>	DATETIME_INTERVAL_CODE PRECISION
<interval type>	DATETIME_INTERVAL_CODE DATETIME_INTERVAL_PRECISION PRECISION

NOTE 38 – The value of the CLI item descriptor area parameter UNNAMED can be derived from the encoded character length of the NAME field.

NOTE 39 – The value of the CLI item descriptor area parameters ALLOC_TYPE, DATA_POINTER and INDICATOR_POINTER are local to the RDA-client environment and are never communicated.

NOTE 40 – The value of the CLI item descriptor area parameter OCTET_LENGTH, and the values pointed at by DATA_POINTER and INDICATOR_POINTER can be derived from the encoding of an RDAValue.

DescriptorValue: the value of the descriptor field identified by DescriptorCode encoded according to the encoding type defined in ISO/IEC 9075-3 (SQL/CLI).

7.3.4 RDARow and RDAValue encoding elements

Function

Encode a row of values

Encoding

```
RDARow                          ::=   SEQUENCE
   {  values                              RDAValueList
   }

RDAValue                        ::=   CHOICE
   {  nullValue                           RDANull,
      character                           RDACharString,
      characterVarying                    RDACharString,
      bit                                 RDABitString,
      bitVarying                          RDABitString,
      smallint                            RDAInteger,
      integer                             RDAInteger,
      decimal                             RDACharString,
      numeric                             RDACharString,
      real                                RDAReal,
      doublePrecision                     RDAReal,
      float                               RDAReal,
      datetime                            RDACharString,
      interval                            RDACharString
   }
```

Encoding Parameters

Values: a list of RDAValues that encode the values of the fields of the row.

NOTE 41 – The encoding of RDARow does not explicitly encode information about the number of values encoded or the data types of the values. This information is provided implicitly in the encoding and by an associated RDAItemDescriptorList.

NullValue: an encoding that indicates the null value.

Datetime: an encoding of a value of type <datetime>.

Interval: an encoding of type <interval>

The other parameters correspond to implementation data types with the same name as defined in ISO/IEC 9075-3 (SQL/CLI). The matching of names for this purpose ignores capitalisation and spaces.

8 Exceptions

The RDA Protocol rules and RDA Operation rules include rules that when evaluated may result in the raising of an exception.

8.1 Exception codes for RDA-specific Conditions

The SQL conditions that may be raised as a result of the invocation of an RDA Operation are listed in Table 12 together with the corresponding status codes.

Table 12 – SQLSTATE class and subclass values for RDA-specific conditions

Condition	Class	Subcondition	Subclass
RDA-specific condition	HZ	attribute not permitted	301
		authentication failure	302
		duplicate request ident	303
		encoding not supported	304
		feature not supported – multiple server transactions	305
		invalid attribute type	306
		invalid fetch count	307
		invalid message type	308
		invalid operation sequence	309
		invalid transaction operation code	310
		mismatch between descriptor and row	311
		no connection handle available	312
		number of values does not match number of item descriptors	313
		transaction cannot commit	314
		transaction state unknown	315
		transport failure	316
		unexpected parameter descriptor	317
		unexpected row descriptor	318
		unexpected rows	319
		version not supported	320
		TCP/IP error	321
		TLS alert	322

8.2 Exception Behaviour

If an RDA generated exception *E* is raised by the RDA-server Operation Rules when evaluating operation *RO* an RDAResponse *R* is constructed with parameters set to the values defined in Table 13 and *R* is regarded as the result of the evaluation of *RO*.

If an RDA generated exception *E* is raised in the RDA-client when evaluating an operation *RO* an RDAResponse *R* is deemed to have been received with parameters as set out in Table 13 and *R* is regarded as the result of the evaluation of *RO*.

NOTE 42 – This may result in a single recursion of the RDAResponse Evaluation rules.

Table 13 – RDAResponse Parameter settings for RDA generated conditions

Parameter	Value
ServerAttributes	An empty list
DynamicFunction	The null string
DynamicFunctionCode	0
More	0
ReturnCode	-1
RowCount	0
CatalogName	The null string
ClassOrigin	‘ISO 9075’
ColumnName	The null string
ConditionNumber	1
ConstraintCatalog	The null string
ConstraintName	The null string
ConstraintSchema	The null string
CursorName	The null string
MessageText	the text of the condition and subcondition from Table 12
NativeCode	0
SchemaName	The null string
ServerName	The null string
SQLState	the SQLState for the condition and subcondition from Table 12
SubclassOrigin	‘ISO 9579’
TableName	The null string
ParameterDescriptor	An empty list
RowDescriptor	An empty list
Rows	An empty list

9 Encodings

The encodings that may be supported by implementations of this International Standard are identified in Table 14. The Definition Subclause for an Encoding defines the encoding of those types used by this International Standard but not otherwise defined.

Table 14 – Codes used to identify TCP/IP encoding

Encoding	Definition Subclause	Encoding Code
The Base encoding	9.1	0
The ASN.1 PER encoding	9.2	1

9.1 The Base Encoding

If a sequence of octets *S* encodes an integer then the bits of *S* taken in a sequence starting with the most significant bit of the first octet of *S* and ending with the least significant bit of the last octet of *S* (or the first octet if *S* has only one octet) are regarded as the bits of the 2's complement binary representation of an integer taken in a sequence starting with the most significant bit. If *S* is empty, it encodes the integer zero.

RDANull: No octets

RDACharacter: A sequence of two octets encoding an integer whose value represents a character *C* of the Basic Multilingual Plane of ISO/IEC 10646 encoded according to UCS-2 as defined in ISO/IEC 10646. The value of the encoding is *C*.

RDAInteger: An octet encoding an integer *L* followed by a sequence *S* of *L* octets encoding an integer *I*. The value of the encoding is *I*. *L* may not be less than zero or greater than eight.

RDAReal: A sequence of eight octets encoding a real number as specified for 64 bit floating point numbers in IEEE 754-1985.

RDAOctetString: A sequence of four octets encoding an integer *L* followed by a sequence *S* of *L* octets. The value of the encoding is the octet string *S*. *L* may not be less than zero.

RDACharString: A sequence of four octets encoding an integer *L* followed by a sequence of *L* encodings of RDACharacter, *CS*. The value of the encoding is *CS*. *L* may not be less than zero.

RDABitString: A sequence of four octets encoding an integer *L* followed by a sequence *S* of *LO* octets where *LO* is the smallest integer greater than or equal to *L* divided by eight. The value of the encoding is the first *L* bits of *S* starting with the least significant bit of the first octet of *S* and ending with the most significant bit of the last octet of *S* (or the first octet if *S* has only one octet). *L* may not be less than zero.

CHOICE OF {*T0, . . . Tn*}: A sequence of one octet encoding an integer *C* followed by an encoding of the *C*th item between the braces, the first item numbered zero. *C* may not be less than zero.

SEQUENCE OF *T*: A sequence of four octets encoding an integer *L* followed by a sequence *S* of *L* encodings of type *T*. *L* may not be less than zero.

SEQUENCE {*T0, . . . Tn*}: The encodings of the items between the braces in sequence.

9.2 The ASN.1 PER Encoding

The encoding shall be according to the ASN.1 encoding rules defined in ISO/IEC 8825-2. The encoding shall use Canonical, Aligned, Packed Encoding Rules (PER).

Where ISO/IEC 8825-2 allows both primitive and constructed encoding, primitive encoding shall be used.

Basic types are encoded by the ASN.1 types:

```
RDANull                 ::=   NULL

RDAInteger              ::=   INTEGER

RDAReal                 ::=   REAL

RDABitString            ::=   BIT STRING

RDAOctetString          ::=   OCTET STRING

RDACharString           ::=   BMPString
```

10 Transport Mappings

A Transport Mapping interprets a Transport Address and makes Transport Facilities provided by a Transport Provider available to RDA.

A Transport Connection may be shared between execution threads. In a multi-threading environment it is the responsibility of the Transport Mapping to ensure that the transmission of one RDA message on a Transport Connection does not interfere with the transmission of another RDA message on the same Transport Connection.

The Transport Mappings defined by this International Standard are identified in Table 15.

Table 15 – Transport Mappings

Transport Mapping	Definition subclause
TCP/IP	10.1
TLS	10.2

NOTE 43 – Future editions of this International Standard and amendments to this edition of this International Standard may define Transport Mappings for other Transport Providers

10.1 Mapping to TCP/IP

TCP/IP is a communication protocol defined by RFC 971 and RFC 793 together. It provides a reliable stream-oriented transport service, which may be used by RDA-clients and RDA-servers for the purpose of RDA communication. Mapping of RDA data transfer requirements to TCP/IP provided facilities are defined in the following clauses.

10.1.1 Transport Address

A Transport Address is represented by a string consisting of a <domain> as defined in RFC 819.

10.1.2 Mapping of Transport Connect

A TCP/IP connection is established as defined in RFC 793 for the OPEN call between the source *S* and the destination *D*. The local socket is set to the IP address corresponding to the Transport Address of *S*, and the foreign socket to the IP address corresponding to the Transport Address of *D*. The port number for an RDA-client is allocated by the RDA-client to be a TCP/IP port number otherwise unused by the client. The port number of all RDA-servers using this Transport Mapping is 630.

10.1.3 Mapping of Transport Disconnect

The Transport Connection is terminated as defined in RFC 793 for the CLOSE call.

10.1.4 Mapping of Transport Fail

The Transport Connection is terminated as defined in RFC 793 for the ABORT call.

10.1.5 Mapping of Transport Send

The possibly concatenated encoding of RDA protocol elements is sent as defined for a buffer in RFC 793 for the SEND call with the PUSH control bit set.

10.1.6 Mapping of Transport Receive

Bytes are read consecutively as defined in RFC 793 for the RECEIVE call until the end of an RDA protocol element is reached.

NOTE 44 – This does not prohibit protocol element concatenation.

10.1.7 Mapping of Transport Errors

If a Transport Connect Request, Transport Disconnect Request, Transport Fail Request, Transport Send or Transport Receive facility fails, then the exceptions are raised: *RDA-specific condition: transport failure* and *RDA-specific condition – TCP/IP error* with the error level and error description placed in the NativeCode and MessageText fields of a status record of the Diagnostic Area

10.1.8 Default Encoding

The Default Encoding for this Transport Mapping is the Base encoding defined in 9.1

10.2 Mapping to TLS

Transport Layer Security (TLS) is a security protocol defined by RFC 2246.

This mapping is the same as that defined in 10.1 except as described below.

10.2.1 Mapping of Transport Connect

It is recommended that the port number of an RDA-server using this Transport Mapping is 630.

> NOTE 45 – This port number is the same as that used by the TCP/IP mapping. Facilities will be provided in future editions of this International Standard to permit determination of whether the TCP/IP or the TLS mapping is used. The port number assigned to this mapping may change at that time.

10.2.2 Mapping of encodings

RDA protocol elements (possibly concatenated) are carried as plain-text fragments in the Transport Layer Security record protocol as defined in RFC 2246.

10.2.3 Mapping of Transport Errors

If an Alert message is received then the exceptions are raised: *RDA-specific condition – transport failure* and *RDA-specific condition: TLS Alert* with the AlertLevel and AlertDescription placed in the NativeCode and MessageText fields of a status record of the Diagnostic Area.

10.2.4 Provision of mandatory security facilities

RDA Transfer Integrity shall be provided by applying Integrity Protection, at least, to all TLS Records carrying RDA Protocol elements.

10.2.5 Provision of optional security facilities

If required, optional security facilities shall be supported as follows:

RDA-server Authentication shall be supported using certificates as defined in ISO/IEC 9594-8.

RDA-client Authentication shall be supported using certificates as defined in ISO/IEC 9594-8.

If the RDA-client represents a single Service User then RDA Client Authentication may also be used to authenticate the RDA Service User as identified by the UserName field of RDAConnect.

RDA Transport Confidentiality shall be supported by using the TLS Record protocol with encryption.

11 Conformance

An implementation shall claim conformance either as an RDA-client or as an RDA-server. A system may contain a number of RDA-clients and a number of RDA-servers.

11.1 RDA-client Conformance

An implementation that claims conformance to this International Standard as an RDA-client shall exhibit external behaviour consistent with having implemented the RDA Operations and RDA Protocol defined in clauses 6 and 7 inasmuch as they relate to an RDA-client, at least one Transport Mapping as defined in clause 10 and the Default Encoding for each Transport Mapping implemented.

An implementation may claim conformance to the RDA-client Application Programming Interface in which case it shall provide a routine interface as defined in Annex B.

An implementation may claim conformance to the RDA Location Server in which case it shall provide the SQL-server defined in Annex D.

Where an implementation claims conformance to this International Standard and also provides an interface as defined in ISO/IEC 9075-3 (SQL/CLI) then the implementation shall exhibit a behaviour consistent with the mapping defined in Annex C.

11.2 RDA-server Conformance

An implementation that claims conformance to this International Standard as an RDA-server shall exhibit external behaviour consistent with having implemented the RDA Operations and RDA Protocol defined in clauses 6 and 7 inasmuch as they relate to an RDA-server, at least one Transport Mapping as defined in clause 10 and the Default Encoding for each Transport Mapping implemented.

An implementation may claim conformance to the RDA-server Application Programming Interface in which case it shall provide a routine interface as defined in Annex B.

An implementation may claim conformance to the RDA Support Server in which case it shall provide the SQL-server defined in Annex E.

11.3 Claims of Conformance

Claims of conformance to this International Standard shall be accompanied by a completed proforma for the information listed in Annex A.

Annex A Conformance Proforma

(normative)

A copy of the proforma in this annex is to be completed for each RDA-client implementation and each RDA-server implementation that claims to conform to this International Standard.

All items in A.1 to A.4 shall be completed and either the items in A.5 or A.6 according to the entry in item A.4.2.

A.1. Identification

A.1.1	Conformance statement for: ISO/IEC 9579:199x, Information Technology — Remote Database Access for SQL (RDA/SQL).
A.1.2	Date of Statement:

A.2. Supplier Details

A.2.1	Organisation:
A.2.2	Contact Name(s):
A.2.3	Address:
A.2.4	Telephone:
A.2.5	Fax:
A.2.6	Email:
A.2.7	Other contact information:

A.3. Implementation Details

A.3.1	Implementation Name:
A.3.2	Version:
A.3.3	Hardware name:
A.3.4	Hardware version:
A.3.5	Operating system name:
A.3.6	Operating system version:
A.3.7	Special configuration requirements:
A.3.8	Other implementation information:

A.4. RDA Support

A.4.1	Does the implementation support the RDA Operations and RDA Protocol (check one):	yes ☐, no ☐
A.4.2	Is this conformance statement for (check one):	RDA-client ☐, RDA-server ☐
A.4.3	Does the implementation also support Edition 2 (check one):	yes ☐, no ☐
A.4.4	Specify a list of amendments implemented (or none):	
A.4.5	Specify a list of technical corrigenda implemented (or none):	
A.4.6	Which Transport Mappings are provided (check all that apply):	TCP/IP ☐, TLS ☐
A.4.7	Which RDA encodings are supported (check all that apply):	Base encoding ☐, ASN.1 PER encoding ☐

A.5. Optional facilities for RDA-clients only

A.5.1	Does the implementation provide RDA/API (check one):	yes ☐, no ☐
A.5.2	If so, which languages (check all that apply)	Ada ☐, C ☐, COBOL ☐, FORTRAN ☐, MUMPS ☐, Pascal ☐, PL/I ☐
A.5.3	Does the implementation provide asynchronous operation (check one):	yes ☐, no ☐
A.5.4	Does the implementation provide message concatenation (check one):	yes ☐, no ☐
A.5.5	Does the implementation provide the RDA Location Server (check one):	yes ☐, no ☐
A.5.6	Does the implementation provide the client resume connection mode (check one):	yes ☐, no ☐
A.5.7	Does the implementation support the server resume connection mode (check one):	yes ☐, no ☐
A.5.8	Which RDA Service User Authentication types are provided (check all that apply):	None ☐, Password ☐, Transfer ☐, AttributeCertificate ☐, Other ☐
A.5.9	What levels of RDA Request Non-repudiation are provided (check all that apply):	None ☐, OriginatorSigned ☐, TtpSigned ☐
A.5.10	What levels of RDA Response Non-repudiation are supported (check all that apply):	None ☐, OriginatorSigned ☐, TtpSigned ☐

A.6. Optional facilities for RDA-servers only

A.6.1	Does the implementation provide RDA/API (check one):	yes ☐, no ☐
A.6.2	If so, which languages (check all that apply)	Ada ☐, C ☐, COBOL ☐, FORTRAN ☐, MUMPS ☐, Pascal ☐, PL/I ☐
A.6.3	Does the implementation provide asynchronous operation (check one):	yes ☐, no ☐
A.6.4	Does the implementation provide message concatenation (check one):	yes ☐, no ☐
A.6.5	Does the implementation provide the RDA Support Server (check one):	yes ☐, no ☐
A.6.6	Does the implementation support the client resume connection mode (check one):	yes ☐, no ☐
A.6.7	Does the implementation provide the server resume connection mode (check one):	yes ☐, no ☐
A.6.8	Which RDA Transaction Co-ordination types are provided (check all that apply):	One-phase ☐, Two-phase ☐
A.6.9	Which RDA Service User Authentication types are provided (check all that apply):	None ☐, Password ☐, Transfer ☐, AttributeCertificate ☐, Other ☐
A.6.10	What levels of RDA Request Non-repudiation are provided (check all that apply):	None ☐, OriginatorSigned ☐, TtpSigned ☐
A.6.11	What levels of RDA Response Non-repudiation are supported (check all that apply):	None ☐, OriginatorSigned ☐, TtpSigned ☐

Annex B RDA Programming Interface

(normative)

This Annex defines an optional RDA Programming Interface, RDA/API.

NOTE 46 – An RDA Programming Interface is not required for conformance to this International Standard. However, if an implementation provides an RDA Programming Interface it must conform to the specification in this Annex.

For those RDA implementations that present an RDA Programming Interface corresponding to RDA/API the Conformance Proforma in the Annex A offers the opportunity to declare such provision.

RDA/API functions provide:

- a means of initiating a Transport Connection,
- a means of identifying a Transport Connection and associating subsequent RDA Operation invocations with it,
- a means of invoking an RDA Operation and retrieving the results of that invocation.

B.1. Notation for defining RDA/API functions

The notation for defining RDA/API functions is that defined in ISO/IEC 9075-3 (SQL/CLI) for defining CLI functions. In addition:

- An RDA/API Parameter *P* of type CHARACTER(*L*) which is an input RDA/API Parameter has an INTEGER or SMALLINT input RDA/API Parameter associated with it whose name is the same as the name of *P* with the addition of the letter "L" at the end which designates the number of characters of the parameter which are significant.
- An RDA/API Parameter *P* of type CHARACTER(*L*) which is an output RDA/API Parameter has an INTEGER or SMALLINT input RDA/API Parameter associated with it whose name is the same as the name of *P* with the addition of the letters "BL" at the end which designates the number of characters of the parameter which are available for RDA/API to write into, and an INTEGER or SMALLINT output RDA/API Parameter associated with it whose name is the same as the name of *P* with the addition of the letter "L" at the end which designates the number of characters of the RDA/API Parameter which are significant when the operation has completed.
- The encoding of a CHARACTER(*L*) parameter is the same as the encoding of the corresponding RDA/API Parameter.

B.2. Mapping RDA/API to a programming language

Mapping an RDA/API function to a programming language is analogous to the mapping defined for SQL/CLI in ISO/IEC 9075-3 (SQL/CLI).

The character set and encoding used in character string parameters is as for the definition of RDACharacter in 9.1.

B.3. Transport Handles

A Transport Handle identifies an SQL-connection. It is allocated by a successfully completed call to the RDAConnect function and is used as an input parameter to other calls which invoke operations on that SQL-connection. A Transport Handle is released by a call to the RDADisconnect function.

NOTE 47 – This parameter is not included in any protocol defined by this International Standard.

B.4. Transport Mapping Codes

The TransportMapping parameter of the RDAConnect function selects the Transport Mapping. The Transport Mapping parameter is a code defined in Table 16.

NOTE 48 – This parameter is not included in any protocol defined by this International Standard.

Table 16 – Transport Mapping Codes

Transport Mapping	Definition subclause	Transport Mapping Code
TCP/IP	10.1	1
TLS	10.2	2

B.5. Transport Connection Management

Transport Connections are established and terminated as required according to the RDA Protocol rules set out in 6.3.

B.6. RDA/API functions

For each RDA Operation defined in clause 7 there is an RDA/API function with the same name. Each such function has parameters, defined in B.8:

- to identify a Transport Connection,
- to provide the encoding parameters of the corresponding RDA Operation,
- to return the protocol parameters of the RDAResponse protocol element.

B.7. RDA/API function invocation

When an RDA/API function *F* is invoked:

1) Let *TH* be the value of the parameter TransportHandle.
2) If *TH* does not identify an allocated Transport Handle, then the function terminates returning invalid handle.
3) The RDA Operation corresponding to *F* is invoked with Encoding Parameters set to the corresponding Function Parameters, using a Transport Connection associated with *TH*.
4) When the RDA Operation has completed the Function Parameters corresponding to RDAResponse Protocol Parameters are set from the corresponding Encoding Parameters.
5) The function terminates returning the value of ReturnCode of the RDADiagnostic Encoding Parameter.

B.8. RDA/API function parameters

```
RDAConnect (
     TransportHandle        OUT   INTEGER,
     TransportMapping       IN    SMALLINT,
     DestinationAddress     IN    CHARACTER(L1),
     DestinationAddressL    IN    SMALLINT,
     DestinationServerName  IN    CHARACTER(L2),
     DestinationServerNameL IN    SMALLINT,
     UserName               IN    CHARACTER(L3),
     UserNameL              IN    SMALLINT,
     AuthenticationType     IN    SMALLINT,
     Authentication         IN    CHARACTER(L4),
     AuthenticationL        IN    SMALLINT,
     MessageContext         IN    CHARACTER(L5),
     MessageContextL        IN    SMALLINT,
     ServerAttributes       OUT   CHARACTER(L6),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L7),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT

RDADisconnect (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L8),
     MessageContextL        IN    SMALLINT,
     ServerAttributes       OUT   CHARACTER(L9),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L10),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT
```

```
RDAEndTran (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L11),
     MessageContextL        IN    SMALLINT,
     CompletionType         IN    SMALLINT
     ServerAttributes       OUT   CHARACTER(L12),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L13),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT

RDAClientAttribute (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L14),
     MessageContextL        IN    SMALLINT,
     ClientAttributes       IN    CHARACTER(L15),
     ClientAttributesL      IN    SMALLINT
     ServerAttributes       OUT   CHARACTER(L16),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L17),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT

RDAStatementPrepare (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L18),
     MessageContextL        IN    SMALLINT,
     StatementIdent         IN    INTEGER,
     StatementText          IN    CHARACTER(L19),
     StatementTextL         IN    INTEGER,
     ServerAttributes       OUT   CHARACTER(L20),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L21),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT,
     ParameterDescriptor    OUT   CHARACTER(L22),
     ParameterDescriptorBL  IN    SMALLINT,
     ParameterDescriptorL   OUT   SMALLINT)
     RowDescriptor          OUT   CHARACTER(L23),
     RowDescriptorBL        IN    SMALLINT,
     RowDescriptorL         OUT   SMALLINT)
     RETURNS SMALLINT
```

```
RDAStatementDeallocate (
     TransportHandle        IN   INTEGER,
     MessageContext         IN   CHARACTER(L24),
     MessageContextL        IN   SMALLINT,
     StatementIdent         IN   INTEGER,
     ServerAttributes       OUT  CHARACTER(L25),
     ServerAttributesBL     IN   SMALLINT,
     ServerAttributesL      OUT  SMALLINT,
     Diagnostics            OUT  CHARACTER(L26),
     DiagnosticsBL          IN   SMALLINT,
     DiagnosticsL           OUT  SMALLINT)
     RETURNS SMALLINT

RDAStatementExecute (
     TransportHandle        IN   INTEGER,
     MessageContext         IN   CHARACTER(L27),
     MessageContextL        IN   SMALLINT,
     StatementIdent         IN   INTEGER,
     ParameterDescriptor    IN   CHARACTER(L28),
     ParameterDescriptorL   IN   SMALLINT
     ParameterData          IN   CHARACTER(L29),
     ParameterDataL         IN   INTEGER
     ServerAttributes       OUT  CHARACTER(L30),
     ServerAttributesBL     IN   SMALLINT,
     ServerAttributesL      OUT  SMALLINT,
     Diagnostics            OUT  CHARACTER(L31),
     DiagnosticsBL          IN   SMALLINT,
     DiagnosticsL           OUT  SMALLINT)
     RETURNS SMALLINT
```

```
RDAStatementExecDirect (
     TransportHandle         IN    INTEGER,
     MessageContext          IN    CHARACTER(L32),
     MessageContextL         IN    SMALLINT,
     StatementIdent          IN    INTEGER,
     StatementText           IN    CHARACTER(L33),
     StatementTextL          IN    INTEGER,
     ParameterData           IN    CHARACTER(L34),
     ParameterDataL          IN    INTEGER,
     ServerAttributes        OUT   CHARACTER(L35),
     ServerAttributesBL      IN    SMALLINT,
     ServerAttributesL       OUT   SMALLINT,
     Diagnostics             OUT   CHARACTER(L36),
     DiagnosticsBL           IN    SMALLINT,
     DiagnosticsL            OUT   SMALLINT,
     ParameterDescriptor     OUT   CHARACTER(L37),
     ParameterDescriptorBL   IN    SMALLINT,
     ParameterDescriptorL    OUT   SMALLINT,
     RowDescriptor           OUT   CHARACTER(L38),
     RowDescriptorBL         IN    SMALLINT,
     RowDescriptorL          OUT   SMALLINT)
     RETURNS SMALLINT

RDAStatementFetchRows (
     TransportHandle         IN    INTEGER,
     MessageContext          IN    CHARACTER(L39),
     MessageContextL         IN    SMALLINT,
     StatementIdent          IN    INTEGER,
     FetchOrientation        IN    SMALLINT,
     FetchOffset             IN    INTEGER,
     FetchCount              IN    INTEGER,
     FetchReturnLimit        IN    INETEGER,
     ServerAttributes        OUT   CHARACTER(L40),
     ServerAttributesBL      IN    SMALLINT,
     ServerAttributesL       OUT   SMALLINT,
     Diagnostics             OUT   CHARACTER(L41),
     DiagnosticsBL           IN    SMALLINT,
     DiagnosticsL            OUT   SMALLINT,
     RowDescriptor           OUT   CHARACTER(L42),
     RowDescriptorBL         IN    SMALLINT,
     RowDescriptorL          OUT   SMALLINT,
     Rows                    OUT   CHARACTER(L43),
     RowsBL                  IN    SMALLINT,
     RowsL                   OUT   SMALLINT)
     RETURNS SMALLINT
```

```
RDAStatementCloseCursor (
     TransportHandle        IN   INTEGER,
     MessageContext         IN   CHARACTER(L44),
     MessageContextL        IN   SMALLINT,
     StatementIdent         IN   INTEGER
     ServerAttributes       OUT  CHARACTER(L45),
     ServerAttributesBL     IN   SMALLINT,
     ServerAttributesL      OUT  SMALLINT,
     Diagnostics            OUT  CHARACTER(L46),
     DiagnosticsBL          IN   SMALLINT,
     DiagnosticsL           OUT  SMALLINT)
     RETURNS SMALLINT

RDAStatementCancel (
     TransportHandle        IN   INTEGER,
     MessageContext         IN   CHARACTER(L47),
     MessageContextL        IN   SMALLINT,
     StatementIdent         IN   INTEGER,
     ServerAttributes       OUT  CHARACTER(L48),
     ServerAttributesBL     IN   SMALLINT,
     ServerAttributesL      OUT  SMALLINT,
     Diagnostics            OUT  CHARACTER(L49),
     DiagnosticsBL          IN   SMALLINT,
     DiagnosticsL           OUT  SMALLINT)
     RETURNS SMALLINT

RDASetCursorName (
     TransportHandle        IN   INTEGER,
     MessageContext         IN   CHARACTER(L50),
     MessageContextL        IN   SMALLINT,
     StatementIdent         IN   INTEGER,
     CursorName             IN   CHARACTER(L51),
     NameLength             IN   SMALLINT,
     ServerAttributes       OUT  CHARACTER(L52),
     ServerAttributesBL     IN   SMALLINT,
     ServerAttributesL      OUT  SMALLINT,
     Diagnostics            OUT  CHARACTER(L53),
     DiagnosticsBL          IN   SMALLINT,
     DiagnosticsL           OUT  SMALLINT)
     RETURNS SMALLINT
```

```
RDAGetCursorName (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L54),
     MessageContextL        IN    SMALLINT,
     StatementIdent         IN    INTEGER,
     ServerAttributes       OUT   CHARACTER(L55),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L56),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT

RDAGetInfo (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L57),
     MessageContextL        IN    SMALLINT,
     StatementIdent         IN    INTEGER,
     InfoType               IN    SMALLINT,
     ServerAttributes       OUT   CHARACTER(L58),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L59),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT

RDAGetTypeInfo (
     TransportHandle        IN    INTEGER,
     MessageContext         IN    CHARACTER(L60),
     MessageContextL        IN    SMALLINT,
     StatementIdent         IN    INTEGER,
     DataType               IN    SMALLINT,
     ServerAttributes       OUT   CHARACTER(L61),
     ServerAttributesBL     IN    SMALLINT,
     ServerAttributesL      OUT   SMALLINT,
     Diagnostics            OUT   CHARACTER(L62),
     DiagnosticsBL          IN    SMALLINT,
     DiagnosticsL           OUT   SMALLINT)
     RETURNS SMALLINT
```

Annex C Mapping of SQL/CLI

(normative)

Where an RDA-client environment provides an SQL/CLI binding as defined in ISO/IEC 9075-3 (SQL/CLI) this International Standard assumes that the RDA-client environment provides a mapping from SQL/CLI functions to RDA Operations that behaves in a way consistent with having implemented the mapping defined in this Annex. This mapping is required for the SQL-environment to conform to ISO/IEC 9075 (SQL).

The SQL/CLI functions are listed in Table C.1. For each SQL/CLI function information is given about whether the function requires the invocation of an RDA Operation, and if so which. In cases where the mapping is other than the immediate invocation of the RDA Operation corresponding to the SQL/CLI function the Table includes a reference to a subsequent subclause that defines the mapping.

Table C.1 – RDA Operations invoked when evaluating an SQL/CLI function

SQL/CLI function	RDA Operations invoked
AllocConnect	No
AllocEnv	No
AllocHandle	No
AllocStmt	No
BindCol	No
BindParam	No
Cancel	RDAStatementCancel
CloseCursor	RDACloseCursor
ColAttribute	No
Connect	RDAConnect
CopyDesc	No
DataSources	No
DescribeCol	No
Disconnect	See C.1
EndTran	See C.2
Error	No
ExecDirect	RDAStatementExecDirect
Execute	RDAStatementExecute
Fetch	RDAStatementFetchRows
FetchScroll	RDAStatementFetchRows
FreeConnect	No
FreeEnv	No
FreeHandle	No
FreeStmt	RDAStatementDeallocate

SQL/CLI function	RDA Operations invoked
GetConnectAttr	No
GetCursorName	RDAGetCursorName
GetData	No
GetDescField	No
GetDescRec	No
GetDiagField	No
GetDiagRec	No
GetEnvAttr	No
GetFunctions	No
GetInfo	RDAGetInfo
GetStmtAttr	No
GetTypeInfo	RDAGetTypeInfo
NumResultCols	No
ParamData	No
Prepare	RDAStatementPrepare
PutData	No
RowCount	No
SetConnectAttr	See C.3
SetCursorName	RDASetCursorName
SetDescField	No
SetDescRec	No
SetEnvAttr	See C.3
SetStmtAttr	See C.3

C.1. SQLDisconnect

When invoking the General Rules of SQLDisconnect, the following General Rule is considered to have been inserted before General Rule 9) of SQLDisconnect:

1) The RDADisconnect Operation is invoked.

2) If there is a Transport Connection, *T*, associated with *C* then that association is terminated and the Transport Disconnect facility may be invoked for *T*.

NOTE 49 – This permits connection re-use.

C.2. SQLEndTran

When invoking the General Rules of SQLEndTran:

The following General Rules and Note are considered to have been inserted before General Rule 12)a) of SQLEndTran within the RDA-client environment:

1) If two-phase commitment is required, then:

   a) If *L1* has more than one element, then the RDAEndTran operation is invoked with CompletionType PREPARE TO COMMIT for each allocated SQL-connection in *L1* requiring two-phase commitment and evalution of these rules continues regardless of any exceptions raised as a result.

   b) Case:

      i) If all the invocations of RDAEndTran in 1)a) succeeded, then RDAEndTran operation is invoked with CompletionType COMMIT for each allocated SQL-connection in *L1*. If any of these invocations of RDAEndTran fail, then the exception is raised: *RDA-specific condition – transaction state unknown.*

      ii) Otherwise, the RDAEndTran operation is invoked with CompletionType ROLLBACK for each allocated SQL-connection in *L1*. If any of these invocations of RDAEndTran fail, then the exception is raised: *RDA-specific condition – transaction state unknown.*

   NOTE 50 – It is implementation defined as to what commitment optimisations may be introduced. For example two-phase commitment need not be invoked for a transaction that was read-only, or for a connection when the operations through that connection were read-only. If commitment optimisations are introduced in this way it is the responsibility of the implementation to ensure that the semantics of ISO/IEC 9075 (SQL) are adequately observed.

General Rule 13)a) of SQLEndTran in the RDA-server environment is considered to have been replaced by:

1) The RDAEndTran operation is invoked with CompletionType ROLLBACK for each allocated SQL-connection in *L1*. If any of these invocations of RDAEndTran fail, then the exception is raised: *RDA-specific condition – transaction state unknown.*

C.3. SQLSetConnectAttr, SQLSetStmtAttr and SQLSetEnvAttr

Following the invocation of SQLSetConnectAttr, SQLSetStmtAttr or SQLSetEnvAttr, subsequent RDA Operation invocations other than the RDAConnect Operation are preceded by invocations of the RDAClientAttribute operation to synchronise the copies of the Attributes in the RDA-server environments with those in the RDA-client environment.

NOTE 51 – Connection and Statement Attributes are associated with a single RDA-server environment. Environment Attributes are associated with all RDA-server environments in the SQL-environment.

C.4. <set transaction statement>

If a <set transaction statement> *S* is prepared for execution using SQLPrepare or SQLExecDirect, then following the execution of *S* using SQLExecute or SQLExecDirect a copy of *S* is prepared and executed for each established SQL-connection and for each SQL-connection established before the next invocation of SQLEndTran in advance of any SQLExecute or SQLExecDirect using that SQL-connection.

Annex D RDA Location Server

(normative)

This Annex presents the schema of an SQL-server, the RDA Location Server, that may be optionally associated with each RDA-client environment that contains information that facilitates heterogeneous interoperation.

For those RDA implementations that provide an RDA Location Server the Conformance Proforma in Annex A offers the opportunity to declare such provision.

D.1. RDA Location Server name and schema

The SQL-server name of the RDA Location Server is RDA_LOCATION_SERVER. This server has a schema named RDA_LOCATION_SCHEMA that contains one table, SERVER_LOCATION.

```
CREATE SCHEMA RDA_LOCATION_SCHEMA
      AUTHORIZATION RDA_DISTRIBUTION_CONTROLLER
```

Each SQL-client maps an SQL-server name known to an application to a specific SQL-server with communication through a specific Transport Provider. In a heterogeneous environment, common mapping information may be shared by different RDA-client environments and by different implementations of component parts of an RDA-client environment. The SERVER_LOCATION table provides mapping information in a standard form.

D.2. Server Location Table

Function

The Server Location Table provides a standard way of accessing server location mapping information.

Definition

```
CREATE TABLE SERVER_LOCATION
   (
      SERVER_NAME                    CHARACTER VARYING (128) NOT NULL,
      TRANSPORT_MAPPING              SMALLINT                NOT NULL,
      TRANSPORT_ADDRESS              CHARACTER VARYING (128) NOT NULL,
      RDA_DESTINATION_SERVER_NAME    CHARACTER VARYING (128) NOT NULL,

      CONSTRAINT PRIMARY KEY (SERVER_NAME)
   )
```

Description

For each SQL-server *S* recorded in SERVER_LOCATION:

SERVER_NAME: the SQL-server name of *S* as known to the Service User.

TRANSPORT_MAPPING: the code identifying the Transport Mapping through which a Transport Connection to *S* can be established as defined in Table 16.

TRANSPORT_ADDRESS: the Transport Address of *S* within the Transport Provider identified by TRANSPORT_MAPPING.

RDA_DESTINATION_SERVER_NAME: the Destination SQL-server name of *S* within the RDA-server environment identified by the values of TRANSPORT_MAPPING and TRANSPORT_ADDRESS taken together.

Annex E RDA Support Server

(normative)

This Annex presents the schema of an optional SQL-server containing information that facilitates heterogeneous interoperation, the RDA Support Server that may be associated with each RDA-server environment.

For those RDA implementations that provide an RDA Support Server the Conformance Proforma in Annex A offers the opportunity to declare such provision.

E.1. RDA Support Server name and schema

Within an RDA-server environment, the Destination SQL-server Name and the SQL-server name of the RDA Support Server is RDA_SUPPORT_SERVER. This server has a schema named RDA_SUPPORT_SCHEMA which contains the tables SERVER_INFORMATION and RDA_REQUEST_LOG.

```
CREATE SCHEMA RDA_SUPPORT_SCHEMA
      AUTHORIZATION RDA_SUPERVISOR
```

E.2. Server Information Table

Function

The Server Information Table has a row for information items of each SQL-server within the RDA-server environment. The rows of this table provide the information returned by the SQLGetInfo function.

Definition

```
CREATE TABLE SERVER_INFORMATION
   (  SERVER_NAME                  CHARACTER(128)         NOT NULL,
      INFO_TYPE                    SMALLINT               NOT NULL,
      INTEGER_INFO_VALUE           INTEGER,
      CHARACTER_INFO_VALUE         CHARACTER VARYING (255),

      CONSTRAINT PRIMARY KEY (SERVER_NAME, INFO_TYPE)
   )
```

Description

SERVER_NAME: the SQL-server name of the SQL-server for which this row provides information.

INFO_TYPE: the code used to identify the information provided by this row. Codes for INFO_TYPE are those defined in Table 25 of ISO/IEC 9075-3 (SQL/CLI).

INTEGER_INFO_VALUE: the value of the information provided by the row if the information provided by this row is defined to have data type INTEGER or SMALLINT by Table 25 of ISO/IEC 9075-3 (SQL/CLI).

CHARACTER_INFO_VALUE: the value of the information provided by the row if the information provided by this row is defined to have data type CHARACTER (*L*) by Table 25 of ISO/IEC 9075-3 (SQL/CLI) for some value of *L*.

E.3. Request Log Table

Function

The Request Log Table has a row for each RDARequest received by the RDA-server environment.

Definition

```
CREATE TABLE RDA_REQUEST_LOG
   (
      TRANSPORT_MAPPING          SMALLINT                   NOT NULL,
      CLIENT_TRANSPORT_ADDRESS   CHARACTER VARYING (128)   NOT NULL,
      REQUEST_PROTOCOL           SMALLINT                   NOT NULL,
      REQUEST_VERSION            SMALLINT                   NOT NULL,
      REQUEST_IDENT              INTEGER                    NOT NULL,
      REQUEST_OPERATION_CODE     SMALLINT                   NOT NULL,
      REQUEST_OPERATION          BIT VARYING (*),
      REQUEST_TIME               TIME                       NOT NULL,
      RESPONSE_TIME              TIME,
      RESPONSE_RETURN_CODE       SMALLINT,
      RESPONSE_DIAGNOSTIC        BIT VARYING (*),

      CONSTRAINT RDA_REQUEST_LOG_PRIMARY_KEY
         PRIMARY KEY (TRANSPORT_MAPPING, CLIENT_TRANSPORT_ADDRESS,
                              REQUEST_IDENT)
   )
```

Description

For a row in RDA_REQUEST_LOG corresponding to an RDARequest *R* and the resulting RDAResponse *S*:

TRANSPORT_MAPPING: the code identifying the Transport Mapping through which *R* was received as defined in Table 16.

CLIENT_TRANSPORT_ADDRESS: the Transport Address of the RDA-client that sent *R*.

REQUEST_PROTOCOL: the RequestProtocol field of *R*.

REQUEST_VERSION: the RequestVersion field of *R*

REQUEST_IDENT: the RequestIdent field of *R*

REQUEST_OPERATION_CODE: the RequestOperationCode field of *R*

REQUEST_OPERATION: the RequestOperation field of *R*

REQUEST_TIME: the time, measured by a clock available to the RDA-server environment, at which *R* was received.

RESPONSE_TIME: the time, measured by the same clock as REQUEST_TIME, at which *S* was sent.

NOTE 52 – It is not required that clocks in different RDA-server environments are synchronised.

RESPONSE_RETURN_CODE: the ReturnCode field of the Diagnostics field of *S*.

RESPONSE_DIAGNOSTIC: the Diagnostics field of *S*.

Annex F Security Service Requirements

(informative)

This Annex presents the requirements for protecting remote database access against potential violations of security. These requirements are described in terms of:

- an identification of the potential violations of security
- requirements for each of: Authentication, Access Control, Transfer Integrity, Transfer Confidentiality, Storage Integrity, Storage Confidentiality and Non-repudiation

Three distinct entities are identified in clause 5.1: Service User, RDA Client, RDA Server. The security service requirements are expressed with reference to these entities in terms of security services similar to those defined in ISO/IEC 7498-2 but adapted to the RDA protocol.

F.1. Potential Vulnerabilities

Communications defined by this International Standard may be vulnerable to the following threats:

Denial of Service: the prevention of authorised access to SQL-data or the delaying of time-critical RDA Operations.

RDA Request Repudiation: the denial by an RDA-client of having sent an RDAMessage requesting an RDA Operation.

RDA Response Repudiation: the denial by an RDA-server of having sent an RDAMessage with the results of an RDA Operation.

RDA-client Masquerade: the pretence by an RDA-client to be some other RDA-client.

RDA-server Masquerade: the pretence by an RDA-server to be some other RDA-server.

Service User Masquerade: the pretence by a Service User to be some other Service User.

Transfer Modification: the unauthorised altering or destruction of an RDAMessage.

Transfer Monitoring: the unauthorised disclosure of the contents of an RDAMessage.

Replay: the unauthorised recording and repetition of an RDAMessage.

The provisions of this International Standard address all these potential violations of security, with the exception of Denial of Service, which is beyond the scope of this International Standard.

F.2. Authentication

The Service User identity may need to be known by the RDA-client environment or the RDA-server environment for the application of identity based access control, for audit purposes, for identification of the "owner" (creator) of data objects or for charging.

To counter Service User masquerade *RDA Service User Authentication* is required.

In situations where the Service User can connect to an RDA-server environment using different RDA clients, or an RDA-client environment includes several Service Users, it may be necessary for the RDA-server environment to identify the RDA-client independently of the Service User identity. For example, the RDA-server may enforce access control based on RDA-client identity to ensure that access to data private to an organisation is not accessible from a system on an external public network, or the RDA-client identity may need to be known for accountability and auditing.

To counter RDA-client masquerade *RDA-client Authentication* is required.

In some cases, such as single user workstations, it may not be necessary to distinguish between the Service User identity and RDA-client identity. In such situations, RDA-client Authentication may be used as the basis for RDA Service User Authentication.

> NOTE 53 – Because of the layering of the security mechanisms and protocol it is not generally possible for SQL Authentication to be used for RDA-client Authentication.

The identity of the RDA-server may need to be confirmed for the Service User to have confidence in the source of data and the effect of other operations. The RDA-server identity may need to be confirmed for the RDA-client environment to apply outgoing access control to inhibit unauthorised access to certain remote SQL-servers.

To counter RDA-server masquerade *RDA-server Authentication* is required.

RDA-client Authentication and *RDA-server Authentication* are forms of Peer Entity Authentication as defined in ISO/IEC 7498-2.

RDA Service User Authentication is corroboration of the Service User identity.

All these forms of authentication can be based on passwords (if there is no risk of monitoring or replaying passwords), cryptographic based authentication tokens, or digital signature mechanisms. These mechanisms can be supported either directly by the RDA Protocol (*RDA Service User Authentication*) or by the Transport Provider (*RDA-client Authentication* and *RDA Server Authentication*).

F.3. Access Control

Protection against attempted unauthorised access to SQL-data may be provided by enforcing access control either:

- at an RDA-client – this generally grants or denies access to an entire SQL-server. This form of access control is termed *RDA Outgoing Access Control*.
- in an intermediate (firewall) system between an RDA-client and an RDA-server – this generally grants or denies specific RDA Operations for a given RDA-server or SQL-server. This form of access control is termed *RDA Firewall Access Control*.
- at an RDA-server – this generally grants or denies access to an entire SQL-server. This form of access control is termed *RDA Incoming Access Control*.
- within an SQL-server – this generally grants or denies access to particular parts of the SQL-data associated with the SQL-server for particular types of operation. This form of access control is termed *SQL Access Control* and is defined in ISO/IEC 9075.

The access control services identified above are forms of access control as defined in ISO/IEC 7498-2.

Access control may be provided using:

- Identity based access control where the entity enforcing the access control uses information on access rights associated with identified entities. The identification generally requires authentication to corroborate the identity.
- Label based access control where access is granted or denied based on the matching of labels associated with protected objects against clearances associated with the accessor.
- Capability based access control where a capability is used to represent right of access.
- Context based access control where contextual information, such as time of day, is used in deciding whether to grant or deny access.

This International Standard currently only specifies use of identity based access control.

RDA Outgoing Access Control applied in an RDA-client environment uses the identity of the Service User to decide whether to grant or deny access to a specific RDA-server. Authentication of the Service User is a local matter. The RDA-server may be authenticated using *RDA Server Authentication*.

RDA Firewall Access Control applied within a Transport Provider uses either the identity of the Service User, optionally authenticated using *RDA User Authentication*, or the identity of the RDA-client, optionally authenticated using *RDA Client Authentication*, to grant or deny a specific RDA Operation to pass through to a specific RDA-server.

RDA Incoming Access Control applied within an RDA-server environment uses the identity of the RDA-client, optionally authenticated using *RDA Client Authentication*, to grant or deny access to an SQL-server within the RDA-server environment.

SQL Access Control applied within an SQL-server uses the identity of the Service User, optionally authenticated using *RDA User Authentication*, to grant or deny access to the SQL-server as defined by ISO/IEC 9075.

The management of access control information (e.g. access control lists) is outside the scope of this International Standard.

F.4. Transfer Integrity

The RDA protocol and SQL data may be subject to Transfer Modification or Transfer Replay attacks when passing over vulnerable data networks.

To protect the integrity of SQL-data against such attacks *RDA Transfer Integrity* is required.

RDA Transfer Integrity is a form of connection integrity with recovery as defined in ISO/IEC 7498-2. Recovery is effected by the protocol rules of 6.3.5 which prescribe that a transaction is rolled-back in the event of communication failure in any component part of the transaction.

RDA Transfer Integrity can be provided using cryptographic check codes or digital signatures applied by the Transport Provider.

F.5. Transfer Confidentiality

RDA Operations and results may be subject to monitoring by unauthorised parties when passing over vulnerable networks.

To protect the confidentiality of RDA Operations and results against such attacks *RDA Transfer Confidentiality* is required.

RDA Transfer Confidentiality is a form of connection confidentiality as defined in ISO/IEC 7498-2.

RDA Transfer Confidentiality can be provided by encipherment mechanisms within the Transport Provider.

F.6. Storage Integrity

The SQL-data associated with an SQL-server may be subject to attempted unauthorised modification.

Access control as described in F.3 may be applied by the SQL-server to protect the integrity of the SQL-data against such attacks. If SQL access control is not sufficiently trusted for integrity, cryptographic mechanisms, such as digital signatures can be used to protect the data or sensitive portions of the data. This protection can be applied by the SQL-client or by the Service User.

> NOTE 54 – The use of cryptographic mechanisms for protecting data may require modification to the schemas of the SQL-data to include the digital signatures.

The use of cryptographic mechanisms for storage integrity is not currently addressed by this International Standard.

Within an SQL transaction a Service User may explicitly connect to several SQL-servers, the RDA-client environment may decompose an SQL-query into fragments that are applied to different SQL-servers or an SQL-server may decompose an SQL-query into fragments that are applied to different SQL-servers. Where data manipulated by a Service User is distributed across several SQL-servers, incomplete or inconsistent changes to SQL-data may occur due to the nature of the SQL operations, system or network failures, or malicious attacks on the integrity of the updates by unauthorised parties.

To protect against such incomplete or inconsistent changes *RDA Transaction Co-ordination* is required.

RDA Transaction Co-ordination can be provided by facilities within the RDA Protocol and RDA Operations, or by other external facilities in conjunction with RDA.

F.7. Storage Confidentiality

The SQL-data associated with an SQL-server may be subject to attempted unauthorised disclosure.

Access control may be applied by the SQL-server to protect the confidentiality of SQL-data against such attacks. If SQL access control is not sufficiently trusted for confidentiality, encipherment, applied by the SQL-client or Service User, can be used to protect the data or sensitive portions of the data.

> NOTE 55 – The use of encipherment may limit the operations that can be performed on the data, in particular with regard to integrity constarints, keys, and selection.

The use of encipherment for storage confidentiality is not currently addressed by this International Standard.

F.8. Non-repudiation

A Service User may deny (repudiate) having initiated use of an SQL server through an RDA operation (or sequence of operations) or an SQL server may repudiate carrying out that operation.

To protect against such attacks *RDA request non-repudiation* and *RDA response non-repudiation* are required.

RDA Request Non-Repudiation and *RDA Response Non-Repudiation* are forms of non-repudiation as defined in ISO/IEC 7498-2.

Non-repudiation can be provided using digital signatures and time-stamping. In some situations, an additional digital signature, provided by a trusted third party against the time-stamped data, may be necessary. This trusted third party signature is only provided if the time-stamp is current.

Annex G Security Profiles

(normative)

This Annex presents profiles that combine the facilities provided by this International Standard in a coherent way to provide four Security Profiles. All but the first provide security services to protect against all threats to RDA security identified in Annex F other than denial of service and repudiation. The four security profiles differ in the management overhead placed on the Service User and the RDA-server environment.

RDA Transaction Co-ordination, RDA Suspend and Resume, Access Control security services and RDA Non-repudiation security services can be applied as an adjunct to any of the security profiles listed below. Access control security services can reduce the risk of Denial of Service attacks.

Security Profile 1: the TCP/IP transport mapping is used with password authentication. This offers a basic level of security without Transfer Integrity or Confidentiality. The RDA-server environment maintains user name and password tables. The Service User maintains a password for each RDA-server environment.

Security Profile 2: the TLS transport mapping is used with password authentication. This offers Transfer Integrity, Transfer Confidentiality and Server Authentication. The RDA-server environment maintains user name and password tables. The Service User maintains a password for each RDA-server environment.

Security Profile 3: the TLS transport mapping is used with Transfer RDA Service User Authentication. The RDA-server environment maintains RDA-client to User Name mapping tables.

Security Profile 4: the TLS transport mapping is used with Attribute Certificate RDA Service User Authentication. Information required for RDA Security is managed on a network basis.

For each Security Profile, Table G.1 indicates the facilities used to achieve that Security Profile and Table G.2 indicates the services provided by that Security Profile.

Table G.1 – Security Profiles – Facilities Used

Facility	***Profile 1***	***Profile 2***	***Profile 3***	***Profile 4***
Transport Mapping	TCP/IP	TLS	TLS	TLS
Authentication	Password	Password	Transfer	Attribute Certificate

Table G.2 – Security Profile – Services Provided

Service	***Profile 1***	***Profile 2***	***Profile 3***	***Profile 4***
RDA Firewall Access Control	Optional	Optional	Optional	Optional
RDA Incoming Access Control	Yes	Yes	Yes	Yes

RDA Outgoing Access Control	Optional	Optional	Optional	Optional
RDA Request Non-repudiation	No	Optional	Optional	Optional
RDA Response Non-repudiation	No	Optional	Optional	Optional
RDA Service User Authentication	Password	Password	Transfer	X.509
RDA Suspend and Resume	Optional	Optional	Optional	Optional
RDA Transaction Co-ordination	Optional	Optional	Optional	Optional
RDA Transfer Confidentiality	No	Yes	Yes	Yes
RDA Transfer Integrity	No	Yes	Yes	Yes
RDA-client Authentication	No	Optional	Optional	Optional
RDA-server Authentication	No	Yes	Yes	Yes
SQL Access Control	Yes	Yes	Yes	Yes

Annex H RDA Operations and Protocol in ASN.1 notation

(informative)

The definition of RDA Operations and Protocol in clauses 6 and 7 are reproduced here for ease of reference.

```
RDAMessage                      ::=  SEQUENCE
   {  messageProtocol                    RDAInteger,
      messageVersion                     RDAInteger,
      messageEncoding                    RDAInteger,
      messageLength                      RDAInteger,
      messageConnectionIdent             RDAInteger,
      messageRequestIdent                RDAInteger,
      messageType                        RDAInteger,
      messageContext                     RDAOctetString,
      messageData                        RDAOctetString,
      messageAuthentication              RDAOctetString
   }
MessageAuthentication           ::=  SEQUENCE
   {  MessageNonRepLevel                 RDAInteger,
      MessageResponseLevel               RDAInteger OPTIONAL,
      MessageTimestamp                   GeneralisedTime,
      OriginatorSignature                MessageAndTimeSig,
      OriginatorCertificate              CertificatePath,
      TtpSignature                       MessageAndTimeSig OPTIONAL,
      TtpCertificate                     CertificatePath OPTIONAL
   }
MessageAndTimeSig               ::=  SIGNATURE
   {  SEQUENCE
      {  MessageProtocol                 RDAInteger,
         MessageVersion                  RDAInteger,
         MessageEncoding                 RDAInteger,
         MessageLength                   RDAInteger,
         MessageRequestIdent             RDAInteger,
         MessageType                     RDAInteger,
         MessageContext                  RDAOctetString,
         MessageData                     RDAOctetString,
         MessageNonRepLevel              RDAInteger,
         MessageResponseLevel            RDAInteger OPTIONAL,
         MessageTimestamp                GeneralisedTime
      }
   }
RDAConnect                      ::=  SEQUENCE
   {  destinationServerName              RDACharString,
      userName                           RDACharString,
      authenticationType                 RDAInteger,
      authentication                     RDAOctetString
   }
RDADisconnect                   ::=  SEQUENCE
   {
   }
```

```
RDAEndTran                      ::=  SEQUENCE
   {  completionType                     RDAInteger
   }

RDAClientAttribute              ::=  SEQUENCE
   {  clientAttributes                   RDAAttributeList
   }

RDAStatementPrepare             ::=  SEQUENCE
   {  statementIdent                     RDAInteger,
      statementText                      RDACharString
   }

RDAStatementDeallocate          ::=  SEQUENCE
   {  statementIdent                     RDAInteger
   }

RDAStatementExecute             ::=  SEQUENCE
   {  statementIdent                     RDAInteger,
      parameterDescriptor                RDAItemDescriptorList,
      parameterData                      RDARowList
   }

RDAStatementExecDirect          ::=  SEQUENCE
   {  statementIdent                     RDAInteger,
      statementText                      RDACharString,
      parameterDescriptor                RDAItemDescriptorList,
      parameterData                      RDARowList
   }

RDAStatementFetchRows           ::=  SEQUENCE
   {  statementIdent                     RDAInteger,
      fetchOrientation                   RDAInteger,
      fetchOffset                        RDAInteger,
      fetchCount                         RDAInteger
   }

RDAStatementCloseCursor         ::=  SEQUENCE
   {  statementIdent                     RDAInteger
   }

RDAStatementCancel              ::=  SEQUENCE
   {  statementIdent                     RDAInteger
   }

RDASetCursorName                    ::=  SEQUENCE
   {  statementIdent                        RDAInteger,
      cursorName                            RDACharString
   }

RDAGetCursorName                    ::=  SEQUENCE
   {  statementIdent                        RDAInteger
   }

RDAGetInfo                          ::=  SEQUENCE
   {  statementIdent                        RDAInteger,
      infoType                              RDAInteger
   }

RDAGetTypeInfo                      ::=  SEQUENCE
   {  statementIdent                        RDAInteger,
      dataType                              RDAInteger
   }
```

```
RDAResponse                     ::=  SEQUENCE
   {  serverAttributes                   RDAAttributeList,
      diagnostics                        RDADiagnostic,
      parameterDescriptor                RDAItemDescriptorList,
      rowDescriptor                      RDAItemDescriptorList,
      rows                               RDARowList
   }

RDAAttributeList                ::=  SEQUENCE OF RDAAttribute

RDAAttribute                    ::=  SEQUENCE
   {  attributeType                      RDAInteger,
      attributeCode                      RDAInteger,
      attributeValue                     RDAInteger,
      statementIdent                     RDAInteger
   }

RDADiagnostic                   ::=  SEQUENCE
   {  dynamicFunction                    RDACharString,
      dynamicFunctionCode                RDAInteger,
      more                               RDAInteger,
      returnCode                         RDAInteger,
      rowCount                           RDAInteger,
      statusRecords                      RDADiagnosticRecordList
   }

RDADiagnosticRecordList         ::=  SEQUENCE OF RDADiagnosticRecord

RDADiagnosticRecord             ::=  SEQUENCE
   {  diagnosticRecords                  RDADiagnosticStatusList
   }

RDADiagnosticStatusList         ::=  SEQUENCE OF RDADiagnosticStatus

RDADiagnosticStatus             ::=  SEQUENCE
   {  diagnosticCode                     RDAInteger,
      diagnosticValue                    RDAValue
   }

RDAItemDescriptor               ::=  SEQUENCE
   {  descriptorEntries                  RDADescriptorEntryList
   }

RDADescriptorEntryList          ::=  SEQUENCE OF RDADescriptorEntry

RDADescriptorEntry              ::=  SEQUENCE
   {  descriptorCode                     RDAInteger,
      descriptorValue                    RDAValue
   }

RDARowList                      ::=  SEQUENCE OF RDARow

RDARow                          ::=  SEQUENCE
   {  values                             RDAValueList
   }

RDAValueList                    ::=  SEQUENCE OF RDAValue
```

```
RDAValue                        ::=  CHOICE
   {  nullValue                          RDANull,
      character                          RDACharString,
      characterVarying                   RDACharString,
      bit                                RDABitString,
      bitVarying                         RDABitString,
      smallint                           RDAInteger,
      integer                            RDAInteger,
      decimal                            RDAInteger,
      numeric                            RDAInteger,
      real                               RDAReal,
      doublePrecision                    RDAReal,
      float                              RDAReal,
      datetime                           RDACharString,
      interval                           RDACharString
   }

RDANull                         ::=  NULL
RDAInteger                      ::=  INTEGER
RDAReal                         ::=  REAL
RDABitString                    ::=  BIT STRING
RDAOctetString                  ::=  OCTET STRING
RDACharString                   ::=  BMPString
```

NOTE 56 – The definitions of RDANull . . . RDACharString depend upon the encoding chosen. The definitions given above are for the ASN.1 PER encoding.

Annex I Encoding of Multiple Rows
(informative)

The ASN.1 encoding defined in this International Standard includes an encoding for the transfer of multiple rows. The ASN.1 fragments that constitute this encoding are reproduced here to facilitate reference to the encoding from other documents.

```
RDAMultipleRows                 ::=  SEQUENCE
    {  rowDescriptor                     RDAItemDescriptorList,
       rows                              RDARowList
    }

RDAItemDescriptor               ::=  SEQUENCE
    {  descriptorEntries                 RDADescriptorEntryList
    }

RDADescriptorEntryList          ::=  SEQUENCE OF RDADescriptorEntry

RDADescriptorEntry              ::=  SEQUENCE
    {  descriptorCode                    RDAInteger,
       descriptorValue                   RDAValue
    }

RDARowList                      ::=  SEQUENCE OF RDARow

RDARow                          ::=  SEQUENCE
    {  values                            RDAValueList
    }

RDAValueList                    ::=  SEQUENCE OF RDAValue

RDAValue                        ::=  CHOICE
    {  nullValue                         RDANull,
       character                         RDACharString,
       characterVarying                  RDACharString,
       bit                               RDABitString,
       bitVarying                        RDABitString,
       smallint                          RDAInteger,
       integer                           RDAInteger,
       decimal                           RDAInteger,
       numeric                           RDAInteger,
       real                              RDAReal,
       doublePrecision                   RDAReal,
       float                             RDAReal,
       datetime                          RDACharString,
       interval                          RDACharString
    }

RDANull                         ::=  NULL
RDAInteger                      ::=  INTEGER
RDAReal                         ::=  REAL
RDABitString                    ::=  BIT STRING
RDAOctetString                  ::=  OCTET STRING
RDACharString                   ::=  BMPString
```

NOTE 57 – The definitions of RDANull . . . RDACharString depend upon the encoding chosen. The definitions given above are for the ASN.1 PER encoding.

ICS 35.060

Price based on 111 pages