RDM Embedded 10.1

RSQL API Reference Manual

Trademarks

Contents

RSQL API Function Reference

There is no passion like that of a functionary for his function.
- Georges Clemenceau

All of the RDMe proprietary SQL C application programming interface functions are described below. While these functions map closely to related functions found in ODBC, they are non-standard but are used by the RDMe ODBC and JDBC libraries.

> Use of this API is only recommended where use of the ODBC functionality is not needed and/or the extra function layer and necessary resources associated with ODBC would have too great a negative impact on the application's target resource requirements.

The RDMe SQL API defines its own enumerator types for data type, status codes, and other needed control constants. These type and constant definitions can be found in files `rsqltypes.h`, `rsqlerrs.h` and `errtab.h` located in the RDMe include directory. Refer to those files for the up-to-date type definitions and constant values but the names of the constants in the following tables will not change.

Table 24. Data Type Identifier Constants of Type SQL_T

SQL_T Constant	Corresponding SQL Data Type
`tNOVAL`	No value has been assigned
`tCHAR`	char
`tVARCHAR`	varchar
`tWCHAR`	wchar
`tWVARCHAR`	wvarchar
`tTINYINT`	tinyint
`tSMALLINT`	smallint
`tINTEGER`	int | integer
`tBIGINT`	bigint
`tFLOAT`	real
`tDOUBLE`	double [precision] | float
`tDATE`	date
`tTIME`	time
`tTIMESTAMP`	timestamp
`tROWID`	rowid
`tCLOB`	char large object | long varchar
`tWCLOB`	wchar large object | long wvarchar
`tBLOB`	binary large object | long varbinary
`tNULL`	value is null

RDMe SQL data values are stored in a general purpose data value container named `RSQL_VALUE` of type struct as shown in the following typedef's from `rsqltypes.h`. SQL data values are passed into or returned by the RDMe SQL API functions use the `RSQL_VALUE` data type so an understanding of how to use this structure is essential. For example, function `rsqlFetch` returns the result set column values for each fetched row in an array of type `RSQL_VALUE`.

Table 25. General Purpose SQL Value Container

```
/* Date, time, and timestamp definitions */
typedef uint32_t DATE_VAL;
typedef uint32_t TIME_VAL;

typedef struct {
    DATE_VAL date;
    TIME_VAL time;
} TIMESTAMP_VAL;

typedef union _value {
    int8_t        tv;
    int16_t       sv;
    int32_t       lv;
    int64_t       llv;
    float         fv;
    double        dv;
    char         *cv;
    void         *pv;
    LONGVAR       lvv;
    DB_ADDR       dba;
    TIMESTAMP_VAL ts;
    wchar_t      *wcv;
} VALUE;

typedef enum _val_status {
    vsOKAY = 0,
    vsTRUNCATE = 1,     /* string truncation or loss of float significance */
    vsDIVBY0,           /* divide by zero */
    vsDOMAIN,           /* argument domain error (e.g., sqrt(-1)) */
    vsOVERFLOW          /* math overflow */
} VAL_STATUS;

/* general purpose SQL data value container */
typedef struct _rsql_value {
    SQL_T      type;    /* internal data type code */
    uint16_t   len;     /* length in bytes of any var-length data or 0 if scalar
*/
    VAL_STATUS status;  /* operation status code */
    VALUE      vt;      /* generic data type container */
} RSQL_VALUE;
```

Function `rsqlGetStmtType` returns a STMT_TYPE argument that identifies the kind of SQL statement that was compiled. These values are given in the following table.

Table 26. Statement Type Identification Constants of Type STMT_TYPE

Constant	Corresponding SQL Statement
`sqlSELECT`	**select** …
`sqlINSERT`	**insert into** *tabname* …
`sqlUPDATE`	**update** tabname …
`sqlOPEN`	**open** *dbname*
`sqlDELETE`	**delete from** *tabname* …

`sqlSTART`	**start transaction** ...
`sqlSAVEPOINT`	**savepoint** ...
`sqlRELEASE`	**release savepoint** ...
`sqlCOMMIT`	**commit transaction**
`sqlROLLBACK`	**rollback transaction**
`sqlCRPROC`	**create procedure** ...
`sqlDRPROC`	**drop procedure** ...
`sqlEXECUTE`	**execute** *procname* ...
`sqlSET`	**set** {**auto commit** | **debug** } ...
`sqlSETCOLUMN`	**set column stats** ...
`sqlLOCK`	**lock table** ...
`sqlUNLOCK`	**unlock table** ...
`sqlIMPORT`	**import** ...
`sqlEXPORT`	**export** ...
`sqlDDL`	SQL DDL statement

All that is needed by an application program to use the RDMe SQL API function is to include header file `rdmsql.h` which includes all of the necessary headers and function prototypes.

The function description pages that follow give the function's prototype, description of each argument, a detailed description of what the function does, an example code snippet, a list of return codes, and a list of related functions. The example code snippet shows the necessary elements in bold-faced text.

Table 1. RDMe SQL API Functions

Function	Description
rsqlAllocConn	Allocate a new connection handle
rsqlAllocStmt	Allocate a new statement handle
rsqlBindParam	Bind a data value to a parameter marker
rsqlCancelRow	Cancel (discard) column value changes to current row
rsqlCloseDB	Close a database
rsqlCloseDBAll	Close all databases that are open on a connection
rsqlCloseStmt	Close the open select statement cursor
rsqlExecDirect	Prepare and execute a SQL statement
rsqlExecProc	Directly execute a pre-compiled SQL stored procedure
rsqlExecute	Execute a compiled SQL statement
rsqlFetch	Fetch the next row of the select statement result set
rsqlFreeConn	Free a connection handle
rsqlFreeStmt	Free a statement handle
rsqlGetAutoCommit	Get the connection handle's current auto commit status
rsqlGetColDescr	Get description information for a select statement result column
rsqlGetConnHandle	Get connection handle associated with specified statement handle
rsqlGetCursorName	Get the cursor name associated for the specified statement handle
rsqlGetData	Get data value for one select statement result column
rsqlGetDateFormat	Get the current date format setting
rsqlGetDateSeparator	Get the current date separator character
rsqlGetDBNames	Get a list of the names of the currently opened databases

Function	Description
rsqlGetDeferBlobMode	Get the current deferred blob reading mode setting
rsqlGetErrorInfo	Get the message associated with the current error code
rsqlGetErrorMsg	Get the message associated with a specific error code
rsqlGetGenCFiles	Get the connection handle's "generate C files" mode
rsqlGetNumParams	Get the number of parameter markers in the compiled statement
rsqlGetNumResultCols	Get the number of result columns in the compiled select statement
rsqlGetParamDescr	Get description information for a SQL statement parameter marker
rsqlGetReadOnlyTrmode	Get the current read only transaction mode
rsqlGetRowCount	Get the count of the # of rows affected by the executed statement
rsqlGetSelectType	Get the type of select statement
rsqlGetStmtType	Get the statement type of the prepared statement
rsqlGetTableName	Get result column's table name
rsqlInitDB	Initialize a database
rsqlLockTables	Issue an explicit lock request for one or more database tables
rsqlMoreResults	Execute next statement in the currently executing stored procedure
rsqlOpenCat	Open a database through its compiled catalog module
rsqlOpenDB	Open a database by name
rsqlPackDate	Pack a CAL_DATE into a binary DATE_VAL
rsqlPackTime	Pack a CAL_TIME into a binary TIME_VAL
rsqlPackTimestamp	Pack a CAL_TIMESTAMP into a binary TIMESTAMP_VAL
rsqlParamData	Check for and initialize rsqlPutData for next data-at-exec parameter
rsqlPrepare	Compile an SQL statement
rsqlPutData	Put a data value for a data-at-exec blob parameter
rsqlRegisterProc	Register a compiled stored procedure
rsqlRegisterUDFs	Register C-based user-defined functions
rsqlRegisterVirtualTables	Register C-based virtual tables
rsqlSetAutoCommit	Set the auto commit status for the specified connection
rsqlSetCursorName	Set the cursor name for the specified statement handle
rsqlSetDateFormat	Set the date constant format for the connection
rsqlSetDateSeparator	Set the current date constant separator character for the connection
rsqlSetDeferBlobMode	Set a statement's deferred reading mode for blob data
rsqlSetErrorCallback	Set an error callback user function
rsqlSetGenCFiles	Set the connection handle's "generate C files" mode
rsqlSetReadOnlyTrmode	Set the current read only transaction mode
rsqlSetTimeout	Set lock wait timeout in seconds for the connection
rsqlTFSInit	Initialize RDMe SQL TFST or TFSS operation
rsqlTFSTerm	Terminate RDMe SQL TFST or TFSS operation
rsqlTransCommit	Commit a transaction
rsqlTransEndReadOnly	End a read only transaction
rsqlTransRelease	Release a transaction savepoint
rsqlTransRollback	Rollback to transaction savepoint or start
rsqlTransSavepoint	Mark a transaction savepoint

Function	Description
rsqlTransStart	Start a transaction
rsqlTransStartReadOnly	Start a read only transaction
rsqlTransStatus	Return the current transaction state for the specified connection
rsqlUnlockTable	Free a read lock on a database table
rsqlUnlockTableAll	Unlock all read locked tables
rsqlUnpackDate	Unpack a binary DATE_VAL into a CAL_DATE structure
rsqlUnpackTime	Unpack a binary TIME_VAL into a CAL_TIME structure
rsqlUnpackTimestamp	Unpack a binary TIMESTAMP_VAL into a CAL_TIMESTAMP structure
rsqlUpdateCol	Update a column value of current row
rsqlUpdateRow	Store the updated column values for the current row

rsqlAllocConn

Allocate a new connection handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlAllocConn(
       HCONN          *pHconn)
```

Arguments

pHconn	(output)	A pointer to the variable to contain the allocated connection handle.

Description

Call `rsqlAllocConn` to allocate a connection handle and open a connection to RDMe SQL. This the first RDMe SQL API function that you need to call (except for, perhaps, `rsqlTFSInit`).

A connection defines a single, thread-safe task. Separate connections can run in separate threads within the same process. After you have called `rsqlAllocConn` you can then open databases, allocate statement handles (`rsqlAllocStmt`), and compile and execute SQL statements.

Example

```
#include "rdmsql.h"
#include "bookshop_cat.h"
    ...
HCONN hdbc;
RSQL_ERRCODE stat;
    ...
    stat = rsqlAllocConn(&hdbc);
    if ( stat == errSUCCESS ) {
        stat = rsqlOpenCat(hdbc, &bookshop_cat, NULL, "s");
        if ( stat != errSUCCESS )
            return stat;
        ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
18	errNOMEMORY	HY013	memory management error

See Also

rsqlAllocStmt

rsqlTFSInit

rsqlAllocStmt

Allocate a new statement handle and associates the statement handle with the connection specified by `hdbc`.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlAllocStmt(
       HCONN          hConn,
       HSTMT         *pHstmt)
```

Arguments

`hConn`	`(input)`	The connection handle.
`pHstmt`	`(output)`	A pointer to the pointer variable to contain the allocated statement handle.

Description

Call `rsqlAllocStmt` to allocate a statement handle to be used to compile and execute SQL statements through the specified connection handle. Multiple statement handles can be allocated for the same connection and can be used independently of each other. In some cases, two statement handles are required in order to perform cursor-based updates and deletes. One handle is needed for the updateable **select** statement and one needed for the **update/delete ... where current of** *cursorname* statement.

> Note that statement handles associated with the same connection handle must be executed within the same thread.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
HSTMT hstmt;
RSQL_ERRCODE stat;
    ...
    stat = rsqlAllocConn(&hdbc);
    ...
    stat = rsqlAllocStmt(hdbc, &hstmt);
    if ( stat != errSUCCESS ) ...
    ...
    stat = rsqlPrepare(hstmt, "select * from customer");
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlAllocConn

rsqlBindParam

Bind a data value to a parameter marker

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlBindParam(
    HSTMT              hStmt,
    uint16_t           parno,
    SQL_T              type,
    const void        *value,
    const int32_t     *pLenValue)
```

Arguments

hStmt	(input)	Statement handle.
parno	(input)	Number of the parameter marker to be bound: 1 <= parno <= # of parameters.
type	(input)	The data type of the parameter value. (See SQL_T Data Type table)
value	(input)	Pointer to the variable containing the value.
pLenValue	(input)	Pointer to variable that contains the length of the parameter value (can be NULL).

Description

Function `rsqlBindParam` is used to associate a host program variable location with a parameter marker that was specified in the SQL statement associated with statement handle `hStmt`. The associated SQL statement must already have been compiled (through a prior call to `rsqlPrepare`) and must contain at least `parno` number of specified parameter markers. Parameters are specified in a SQL statement string using "?" symbols and are numbered in left-to-right order beginning with 1. The data type specified by type must be compatible with the use of the parameter in the statement but the type check may not occur until the statement is executed (`rsqlExecute`). So, `errPARMTYPE` can be returned by `rsqlBindParam` to indicate an invalid parameter value data type, the same error code may not be returned until the call to `rsqlExecute`.

The `pLenValue` argument is a pointer to an `int32_t` variable that is used to specify the length of any non-null-terminated variable length parameter types (e.g., `tBINARY`). It is also used to specify a null parameter value (`*pLenValue = -1`) and to specify a data-at-exec parameter (`*pLenValue = -2`).

For data-at-exec parameters the value pointer argument is not used to point to the parameter value itself (this is done through calls to `rsqlPutData`) but will be returned to the application when function `rsqlParamData` is called to begin processing of the data-at-exec parameter,

Note: Data-at-exec parameters can only be specified for blob data types.

Note: The `rsqlBindParam` does not support binding to single character data. All character data must be bound to NULL terminated strings.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
uint16_t nopars, nocols;
RSQL_VALUE *row;
char        bio_search[50] = "%American%";

    ...
    stat = rsqlPrepare(hstmt, "select * from author where short_bio = ?");
    if ( stat == errSUCCESS ) {
        stat = rsqlBindParam(hstmt, 1, tSMALLINT, &bio_search, NULL);
        if ( stat == errSUCCESS ) {
            stat = rsqlExecute(hstmt);
            while ( stat == errSUCCESS )
                stat = rsqlFetch(hstmt, &row);
        }
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
28	errINVARG	HY009	invalid argument value
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state
117	errPARMTYPE	HY105	invalid parameter type
142	errDATAATEXEC	RX021	data-at-exec params only allowed with INSERT VALUES/UPDATE

143	errBLOBPARSONLY	RX022	data-at-exec params only allowed for blob (long var...) columns

See Also

rsqlParamData

rsqlPutData

rsqlCancelRow

Cancel (discard) the updated column values for the current row

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlCancelRow(
    HSTMT              hStmt)
```

Arguments

hStmt	(input)	Statement handle associated with the **select** statement whose rows are currently being fetched (rsqlFetch).

Description

Function rsqlCancelRow can be called to discard the updated column values made by prior calls to rsqlUpdateCol for the **select** statement result set whose rows are currently being fetched via calls to rsqlFetch.

Only the changes made by calls to rsqlUpdateCol since the last call to rsqlUpdateRow (if there was one) are canceled. The changes stored by prior calls to rsqlUpdateRow remain intact.

Example

```
#include "rdmsql.h"

main()
{
    HCONN hdbc;
    HSTMT hstmt;
    RSQL_ERRCODE stat;
    RSQL_VALUE *row, newcomm;


    rsqlAllocConn(&hdbc);
    rsqlOpenDB(hdbc, "bookshop", "s");
    rsqlAllocStmt(hdbc, &hstmt);

    newcomm.type = tDOUBLE;
    newcomm.len = 0;
    rsqlExecDirect(hstmt, "start transaction");
    rsqlExecDirect(hstmt, "select mgrid, commission from acctmgr for update");

    while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
        if ( row[1].vt.dv >= 0.03 ) {
```

```
            printf("Change commission for %s from %.2f ",
                                    row[0].vt.cv, row[1].vt.dv);
            if ( row[1].vt.dv >= 0.05 )
                newcomm.vt.dv = row[1].vt.dv + 0.05;
            else if ( row[1].vt.dv >= 0.03 )
                newcomm.vt.dv = row[1].vt.dv + 0.04;
            else
                newcomm.vt.dv = row[1].vt.dv + 0.01;
            printf("to %.2f?\n", newcomm.vt.dv);
            rsqlUpdateCol(hstmt, 2, &newcomm);
            if ( gets(reply) && (reply[0] == 'y' || reply[0] == 'Y') )
                stat = rsqlUpdateRow(hstmt);
            else
                stat = rsqlCancelRow(hstmt);

            if ( stat != errSUCCESS )
                printf("problem updating %s\n", row[0].vt.cv);
        }
    }
    rsqlExecDirect(hstmt, "commit");
}
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlUpdateCol

rsqlUpdateRow

rsqlCloseDB

Close a database

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlCloseDB(
    HCONN               hConn,
    const char         *dbname)
```

Arguments

hConn	(input)	Connection handle.
dbname	(input)	A string specifying the name of the database to be closed.

Description

Call `rsqlCloseDB` to close the database specified in the `dbname` argument string. Databases are opened either through calls to functions `rsqlOpenDB` or `rsqlOpenCat` or through execution of the **open database** statement. All databases that are opened on a connection are closed when `rsqlFreeConn` is called.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
        ...
stat = rsqlAllocConn(&hdbc);
if ( stat == errSUCCESS ) {
```

```
        stat = rsqlOpenDB(hdbc, "bookshop@Srvr1:21553;nsfawards@Srvr2:21555", "s");
        if ( stat != errSUCCESS ) {
                printf("unable to open the databases\n");
                ...
        }
        ... access the databases
        rsqlCloseDB(hdbc, "bookshop");
        ... access only nsfawards
...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
41	errDBNOTOPEN	42000	database not open
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlOpenCat

rsqlOpenDB

rsqlCloseDBAll

rsqlFreeConn

rsqlCloseDBAll

Close all open database on connection.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlCloseDBAll(
    HCONN               hConn)
```

Arguments

`hConn`	`(input)`	Connection handle.

Description

Call `rsqlCloseDBAll` to close all of the databases that are open on connection `hConn`. Databases are opened either through calls to functions `rsqlOpenDB` or `rsqlOpenCat` or through execution of the **open database** statement. Note that all databases that are opened on a connection are also automatically closed when `rsqlFreeConn` is called.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
        ...
stat = rsqlAllocConn(&hdbc);
if ( stat == errSUCCESS ) {
       stat = rsqlOpenDB(hdbc, "bookshop@Srvr1:21553;nsfawards@Srvr2:21555", "s");
```

```
        if ( stat != errSUCCESS ) {
                printf("unable to open the databases\n");
                ...
        }
        ... access the databases
        rsqlCloseDBAll(hdbc);
...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlOpenCat

rsqlOpenDB

rsqlCloseDB

rsqlFreeConn

rsqlCloseStmt

Close or free a statement handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlCloseStmt(
    HSTMT             hStmt)
```

Arguments

`hStmt`	`(input)`	Statement handle.

Description

Function `rsqlCloseStmt` is used to close a statement handle. The associated statement is returned to the prepared (compiled) but not executed state.

Example

```
#include "rdmsql.h"
    ...
static void DisplayResult(void *, RSQL_VALUE *);
static void HandleError(void *, RSQL_ERRCODE);
    ...
HCONN hdbc;
HSTMT hstmt;
RSQL_ERRCODE stat;
    ...
    stat = rsqlAllocStmt(hdbc, &hstmt);
    ...
    stat = rsqlPrepare(hstmt, "select * from salesperson");
    if ( stat == errSUCCESS ) {
       stat = rsqlExecute(hstmt);
       if ( stat == errSUCCESS ) {
           while ( (stat = rsqlFetch(hstmt, row)) == errSUCCESS ) {
               DisplayResult(hstmt, row);
               if ( user_cancelled ) {
                   stat = rsqlCloseStmt(hstmt);
                   break;
               }
           }
       }
    }
    if ( stat != errSUCCESS ) HandleError(hstmt, stat);
```

```
...
stat = rsqlFreeStmt(hstmt);
...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlAllocStmt

rsqlFreeStmt

rsqlExecDirect

Prepare and execute an SQL statement

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlExecDirect(
    HSTMT              hStmt,
    const char        *sqlstr)
```

Arguments

hStmt	(input)	Statement handle.
sqlstr	(input)	A pointer to the character string containing the SQL statement to be compiled and executed.

Description

You can call `rsqlExecDirect` in order to prepare (i.e., compile the statement) and execute a RDMe SQL statement. Status code errSUCCESS is returned when the specified statement has compiled and executed without error. Otherwise, an error code that identifies the error will be returned.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE *row;
uint16_t    numcols;
       ...
    stat = rsqlExecDirect(hstmt, "select * from author");
    if ( stat == errSUCCESS ) {
       while ( stat == errSUCCESS ) {
           stat = rsqlFetch(hstmt, &row, &numcols);
           DisplayResultRow(row, numcols);
       }
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
45	errNUMPAR	07002	insufficient number of parameters specified

Many other error codes

See Also

rsqlPrepare

rsqlExecute

rsqlExecProc

Directly execute a pre-compiled SQL stored procedure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlExecProc(
    HTMT               hStmt,
    const char        *name,
    uint16_t           noargs,
    const RSQL_VALUE  *args)
```

Arguments

hStmt	(input)	Statement handle.
name	(input)	A pointer to a string containing the name of the stored procedure to execute.
noargs	(input)	The number of procedure arguments in the `args` array.
args	(input)	A pointer an `RSQL_VALUE` array containing the procedure argument values.

Description

Call `rsqlExecProc` to directly execute a pre-compiled stored procedure. Argument name is a string containing the name of the stored procedure to be executed. The noargs value must be equal to the number of arguments specified in the **create procedure** statement. Each entry in the args array must contain a value of the correct type for the corresponding stored procedure argument. The specified stored procedure must have been previously registered through a call to `rsqlRegisterProc`.

A successful return from a call to `rsqlExecProc` behaves just as if an **execute** statement on the specified stored procedure has been executed. Where a retrieval stored procedure contains more than one **select** statement calls to `rsqlMoreResults` will execute each subsequent statement returning status `errNOMOREDATA` when there are no more statements to execute.

Use of this function allows all SQL database access to be defined in stored procedures so that an application program can be created that does not need to load the SQL compilation component thus using only the RDMe SQL execution engine.

Example

The declaration for the stored procedure used in this example is given below.

```
create procedure get_sales_team(mgr char) as
       select sale_name manager from salesperson where sale_id = mgr
       select * from salesperson where mgr_id = mgr
end proc;
```

The code example below shows how this procedure is registered and directly executed.

```
#include "rdmsql.h"
#include "get_sales_team_ssp.h"
       ...
HSTMT hstmt, *hdbc;
RSQL_ERRCODE stat;
RSQL_VALUE mgr;
       ...
       stat = rsqlRegisterProc(hdbc, &get_sales_team_defn);
       ...
       mgr.type = tCHAR;
       mgr.vt.cv = "BNF";
       stat = rsqlExecProc(hstmt, "get_sales_team", 1, &mgr);
       while ( stat == errSUCCESS ) {
               DisplayResultSet(hstmt);
               stat = rsqlMoreResults(hstmt);
       }
       if ( stat != errNOMOREDATA ) ErrorHandler(hstmt, stat);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
72	errNOPROC	RX009	stored procedure file not found
73	errNUMARGS	21000	invalid number of arguments specified

See Also

rsqRegisterProc

rsqlExecute

Execute a compiled SQL statement

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlExecute(
    HSTMT              hStmt)
```

Arguments

`hStmt`	`(input)`	Statement handle.

Description

Call `rsqlExecute` to execute the compiled statement associated with statement handle `hStmt`. It is possible that the type checking of some bound parameter values needed to be deferred so that any that are invalid at the time of this call will result in status code `errPARMTYPE` being returned.

If any data-at-exec blob type (`tBLOB`, `tCLOB`, `tWCLOB`) were bound to one or more of the statement's parameter markers then `rsqlExecute` will return status `errNEEDDATA` indicating that those parameter values are to be supplied through subsequent calls to `rsqlParamData` and `rsqlPutData`.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
struct acctmgr {
    char   mgr_id[8];
    char   name[25];
    double comm;
} mgr;
    ...
    stat = rsqlPrepare(hstmt, "insert into acctmgr values ?, ?, current_date(),
?");
    if ( stat != errSUCCESS ) HandleError(hstmt, stat);
    rsqlBindParam(hstmt, 1, tCHAR, &mgr.mgr_id, NULL);
    rsqlBindParam(hstmt, 2, tCHAR, &mgr.name, NULL);
    rsqlBindParam(hstmt, 3, tDOUBLE, &mgr.comm, NULL);
    while ( GetOutletData(&mgr) == TRUE ) {
       stat = rsqlExecute(hstmt);
           if ( stat != errSUCCESS ) {
               ...
```

```
                break;
            }
        }
        ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-4	errNEEDDATA	01000	need data
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
41	errDBNOTOPEN	42000	database not open
45	errNUMPAR	07002	insufficient number of parameters specified
65	errINVSTATE	RX008	invalid statement state
117	errPARMTYPE	HY105	invalid parameter type

See Also

rsqlPrepare

rsqlBindParam

rsqlParamData

rsqlPutData

rsqlFetch

Fetch the next row of the **select** statement result set

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlFetch(
    HSTMT               hStmt,
    const RSQL_VALUE **pResult,
    uint16_t           *pNoCols)
```

Arguments

hStmt	(input)	Statement handle.
pResult	(output)	A pointer to a RSQL_VALUE pointer that will contain the address of the array of result column values associated with the fetched row.
pNoCols	(output)	A pointer to a uint16_t integer variable that will contain the number of columns in the result set. This is the same value that is returned by function rsqlGet-NumResultCols.

Description

Function rsqlFetch is called to retrieve the next row in the result set for the select statement associated with hstmt that has been compiled (rsqlPrepare) and executed (rsqlExecute). The column result values are returned through the returned pResult argument pointer which is an array of *pNoCols RSQL_VALUE entries each containing the value of its corresponding select result column. The data contained in this array must be copied out if it needs to survive because any pointers to variable length data (e.g., tCHAR strings) are reused by SQL in subsequent calls to rsqlFetch.

Access to the column values in the RSQL_VALUE container is described in the SQL Data Types section of the SQL User Guide.

A NULL can be specified for either pResult or pNoCols. Passing a NULL for pNoCols simply means that the calling program is not interested in that value (which is the same as that which would be returned by a call to rsqlGetNumResultCols). If pResult is NULL then the column values must be retrieved through calls to rsqlGetData.

Function rsqlCloseStmt must be called if it is necessary to terminate fetching the rows of the result set through calls to rsqlFetch before it has completed (i.e., returned status errNOMOREDATA).

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
```

```
RSQL_ERRCODE stat;
const RSQL_VALUE  *row;
uint16_t     nocols;
int16_t      region = 1;
    ...
    stat = rsqlPrepare(hstmt, "select * from salesperson where region = ?");
    if ( stat == errSUCCESS ) {
        stat = rsqlBindParam(hstmt, 1, tSMALLINT, &region, NULL);
        if ( stat == errSUCCESS ) {
            stat = rsqlExecute(hstmt);
            while ( stat == errSUCCESS )
                stat = rsqlFetch(hstmt, &row, &nocols);
        }
    }
    ...
```

SQL User's Guide

Retrieving Select Statement Results

SQL Data Types and Values (RSQL_VALUE)

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-1	errNOMOREDATA	02000	no more data
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
53	errNOTSELECT	24000	current statement is not SELECT
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlGetData

rsqlFreeConn

Free a connection handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlFreeConn(
    HCONN      hConn)
```

Arguments

hConn	(input)	Connection handle.

Description

Call `rsqlFreeConn` to close a connection and free the connection handle and all statement handles associated with the connection handle.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
RSQL_ERRCODE stat;
    ...
    stat = rsqlAllocConn(&hdbc, 0);
    ...
    ...         do SQL on this connection
    ...
    stat = rsqlFreeConn(hdbc);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlAllocConn

rsqlFreeStmt

Free a statement handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlFreeStmt(
    HSTMT             hStmt)
```

Arguments

`hStmt`	`(input)`	Statement handle.

Description

Function `rsqlFreeStmt` is used to free a statement handle. An open cursor will be closed and all of the memory allocated on the statement handle will be freed.

Example

```
#include "rdmsql.h"
    ...
static void DisplayResult(void *, RSQL_VALUE *);
static void HandleError(void *, RSQL_ERRCODE);
    ...
HCONN hdbc;
HSTMT hstmt;
RSQL_ERRCODE stat;
    ...
    stat = rsqlAllocStmt(hdbc, &hstmt);
    ...
    stat = rsqlPrepare(hstmt, "select * from salesperson");
    if ( stat == errSUCCESS ) {
       stat = rsqlExecute(hstmt);
       if ( stat == errSUCCESS ) {
           while ( (stat = rsqlFetch(hstmt, row)) == errSUCCESS ) {
               DisplayResult(hstmt, row);
               if ( user_cancelled ) {
                   stat = rsqlCloseStmt(hstmt);
                   break;
               }
           }
       }
    }
    if ( stat != errSUCCESS ) HandleError(hstmt, stat);
```

```
...
stat = rsqlFreeStmt(hstmt);
...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlAllocStmt

rsqlCloseStmt

rsqlGetAutoCommit

Get the connection handle's current auto commit status

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetAutoCommit(
    const HCONN    hConn,
    int16_t       *pAutocommit)
```

Arguments

`hConn`	(input)	Connection handle.
`pAutocommit`	(input)	A pointer to an `int16_t` variable to contain the auto commit state: 1 = on, 0 = off.

Description

Call `rsqlGetAutoCommit` to determine the auto commit mode status for connection `hConn`. The integer variable pointed to by `pAutocommit` will be 1 when auto commit mode is enabled and 0 when it is not enabled. When auto commit is enabled, each SQL **insert**, **update**, and **delete** statement is automatically performed within its own transaction. Note that when explicit locking (e.g., **lock table** statement) is not allowed when auto commit mode is enabled.

It is highly recommended that you do not use auto commit mode. It is very important that all related database modifications be performed within explicit transactions (e.g., through calls to `rsqlTransStart` and `rsqlTransCommit`) in order to maintain the logical consistency (based on the application's requirements) of the database.

Auto commit mode is initially disabled in the RDMe SQL native API. Note that it is initially enabled in the RDME SQL ODBC/JDBC API.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
int16_t auto_commit_flag;
    ...
    /* turn off auto commit mode if enabled */
    rsqlGetAutoCommit(&auto_commit_flag, hdbc);
    if ( auto_commit_flag )
        rsqlSetAutoCommit(hdbc, 0);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlSetAutoCommit

rsqlTransStart

rsqlGetColDescr

Get description information for a specific select statement result column

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetColDescr(
    HSTMT              hStmt,
    int16_t            colno,
    char              *colname,
    uint16_t           lenColname,
    uint16_t          *pLenColname,
    SQL_T             *pType,
    int16_t           *pWidth,
    int16_t           *pPrec,
    int16_t           *pScale,
    int16_t           *pNullable)
```

Arguments

hStmt	(input)	Statement handle.
colno	(input)	The ordinal number (1..# of columns) of the result column.
colname	(output)	A pointer to the char array buffer to contain the column name (can be NULL).
lenColname	(input)	The length of the colname buffer (including null terminator).
pLenColname	(output)	A pointer to the variable that will contain the actual colname length (including null terminator) (can be NULL).
pType	(output)	A pointer to SQL_T variable to contain the column's data type (can be NULL).
pWidth	(output)	A pointer to the variable to contain the display width (can be NULL).
pPrec	(output)	A pointer to the variable to contain the precision (can be NULL).
pScale	(output)	A pointer to the variable to contain the scale (unused) (can be NULL).
pNullable	(output)	A pointer to the variable that indicates that the column can be assigned a null (can be NULL).

Description

Issue a call to rsqlGetColDescr for each column of a previously compiled **select** statement in order to get a description of the column. The description information that is returned by the function is defined in the following table.

Table 29. rsqlGetColDescr Output Arguments

Output Argument	Result Value
colname	The name of the column. For expressions, this either contains the expression string or the alias name, if specified.
pLenColname	The total length of the column name/heading (includes null terminator).
pType	The column result's data type.

pWidth	The maximum display width.
pPrec	The precision of the result value (i.e., number of significant digits)
pScale	The scale (number of decimal places, currently unused)
pNullable	Indication as to whether the column can be assigned a null value: 1 => nullable, 0 => not nullable or unknown. Only applies on **select** statements that are updateable.

You can call first `rsqlGetColDescr` with a null `colname` argument to return the column name's string length in the `pLenColname` argument if you want to dynamically allocate the `colname` buffer. However, the maximum length of a column (or table or database) name is given by the `NAMELEN #define` that is contained in the standard RDMe SQL header file "rdmsql.h".

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
uint16_t nocols;
struct col_descr_pkt {
    char     *name;
    int16_t   buflen;
    SQL_T     type;
    int16_t   width;
    int16_t   nullable;
} *ColDescrs;
    ...
    stat = rsqlPrepare(hstmt, "select * from salesperson", NULL, NULL, &nocols);
    if ( stat == errSUCCESS ) {
       ColDescrs = malloc(nocols*sizeof(struct col_descr_pkt));
       for ( cno = 0; cno < nocols; ++cno ) {
               /* fetch size of column name/heading */
               rsqlGetColDescr(hstmt, cno+1, NULL, 0, &ColDescrs[cno].buflen,
                               NULL, NULL, NULL, NULL, NULL);

               /* allocate column name buffer */
               ColDescrs[cno].name = malloc(clen+1);

               /* fetch all column descriptor info */
               rsqlGetColDescr(hstmt, cno+1, ColDescrs[cno].name, ColDescrs[cno].buflen,
                       NULL, &ColDescrs[cno].type, &ColDescrs[cno].width,
                       NULL, NULL, &ColDescrs[cno].nullable);
       }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-2	errSQLERROR	RX002	internal SQL error
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value
33	errCOLNUMBER	07009	invalid descriptor index (column number)
53	errNOTSELECT	24000	current statement is not SELECT
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlPrepare

rsqlGetNumResultCols

rsqlGetConnHandle

Get the connection handle on which a statement handle is associated

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetConnHandle(
       HSTMT           hStmt)
       HCONN          *pHconn)
```

Arguments

hStmt	(input)	Statement handle.
pHconn	(output)	A pointer to the variable to contain the associated connection handle.

Description

Function `rsqlGetConnHandle` can be called to return the connection handle on which the specified statement handle, `hStmt`, has been allocated. The connection handle is returned in the `HCONN` variable pointed to by argument pHconn.

Example

```
#include "rdmsql.h"
...
static TRANS_STAT StmtTransStatus(
       HSTMT     hstmt)
{
    HCONN      hdbc;
    TRANS_STAT trstat;

    rsqlGetConnHandle(hstmt, &hdbc);

    rsqlTransStatus(hdbc, &trstat);

    return trstat;
}
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlAllocConn

rsqlAllocStmt

rsqlGetCursorName

Get the cursor name associated for the specified statement handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetCursorName(
    HSTMT       hStmt,
    char       *cursor,
    uint16_t    lenCursor,
    uint16_t   *pLenCursor)
```

Arguments

`hStmt`	`(input)`	Statement handle.
`cursor`	`(output)`	A pointer to a char array into which the cursor name is copied (can be NULL).
`lenCursor`	`(input)`	The maximum size of the char array (including null terminator).
`pLenCursor`	`(output)`	The actual size of the cursor name string (including null terminator).

Description

Call function `rsqlGetCursorName` to retrieve the current cursor name for the updateable **select** statement that has been compiled under statement handle `hstmt`. The returned cursor name can be used with a positioned **delete** or **update** statement to make modifications to the current row of the **select** statement. The maximum size of a cursor name (including null terminator) is `NAMELEN`, a `#define` constant contained in the standard RDMe SQL header file: "rdmsql.h".

If `cursor` is NULL, `pLenCursor` will still return the total number of characters (including the null-termination character for character data) available to return in the buffer pointed to by `cursor`.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt1, hstmt2;
char  cursor[NAMELEN];
char  stmt[80];
    ...
    rsqlPrepare(hstmt1, "select * from patron", NULL, NULL, NULL);
    rsqlGetCursorName(hstmt1, cursor, NAMELEN, NULL);
    sprintf(stmt, "delete from patron where current of %s", cursor);
    rsqlPrepare(hstmt2, stmt, NULL, NULL, NULL);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-5	errINVHANDLE	02002	invalid connection or statement handle
35	errFCNSEQ	HY010	function call sequence error

See Also

rsqlSetCursorName

rsqlGetData

Get the value for one of the **select** statement result set columns

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetData(
    HSTMT               hStmt,
    uint16_t            colno,
    const RSQL_VALUE **pResult,
    uint32_t            maxlen,
    uint32_t           *pRemlen)
```

Arguments

`hStmt`	`(input)`	Statement handle associated with the select statement whose rows are currently being fetched (`rsqlFetch`).
`colno`	`(input)`	The number of the result set column whose value is to be returned (beginning at 1).
`pResult`	`(output)`	A pointer to a `RSQL_VALUE` pointer variable into which a pointer to the column result value will be returned.
`maxlen`	`(input)`	The maximum size in bytes of the amount of blob data to be read.
`pRemlen`	`(output)`	The amount of blob data that was remains to be read at the time the function is called.

Description

Function `rsqlGetData` can be used to return the currently fetched value of any column from the **select** statement result set whose rows are currently being fetched via calls to `rsqlFetch`. While it can be called for columns of any data type it is primarily designed to be used to retrieve blob (**long varchar**, **long wvarchar**, **long varbinary**) data values which may require multiple calls to `rsqlGetData` in order to retrieve particularly large blob data values.

The **select** statement result column whose data is to be returned is specified by the column number, `colno` which must be a value between 1 and the number of result columns in the **select** statement.

The `pResult` argument is a pointer to a `RSQL_VALUE` pointer variable that will contain a pointer to the result value container. Access to the actual data value is through this container based on its result data type. For example, for blob data current access data is found in the `*pResult->vt.lvv LONGVAR struct` where `*pResult->vt.lvv.buf` is a pointer to the blob data value of number of bytes. Note that for **long varchar** or **long wvarchar** blob data, the entire blob value is considered to be a single string so that the null terminator will only be the last byte returned by the final call to `rsqlGetData`.

The `maxlen` argument specifies the maximum number of bytes of blob data to be returned at a time. An application would typically allocate a buffer of size `maxlen` and then after calling `rsqlGetData` do a `memcpy` of `pResult->len` bytes from `*result->vt.lvv.buf` into the allocated buffer.

The amount of blob data that remains to be read at the time of the call is output to the `uint32_t` variable pointed to by the `pRemlen` argument. This allows you to make an initial call `rsqlGetData` passing a `NULL` for `pResult` but with a valid `pRemlen` argument in order to discover the actual size of the blob data that can guide whether or not you need to fetch the blob in blocks.

When making multiple calls to `rsqlGetData` to retrieve large blob data values a "chunk" at a time you should continue the function calls, processing `pResult->len` amount of data each time, until either `rsqlGetData` returns status `errNOMOREDATA` or `*Remlen = 0`.

Access to the column value in the RSQL_VALUE container is described in the SQL Data Types section of the SQL User Guide.

Example

```
#include "rdmsql.h"
    ...
HSTMT        hstmt;
    ...

static void DisplayAuthors()
{
    RSQL_RESULT *val;
    char         biobuf[66];
    uint32_t     len_left;

    rsqlExecDirect(hstmt, "select full_name, short_bio from author");
    while ( rsqlFetch(hstmt, NULL, NULL) == errSUCCESS ) {
        /* get and display author's full name string */
        rsqlGetData(hstmt, 1, &val, 0, NULL);
        printf("Author name: %s\nShort bio  : ", val->vt.cv);

        /* get author's bio a chunk at a time */
        while ( rsqlGetData(hstmt, 2, &val, 65, &len_left) == errSUCCESS ) {
            if ( val->type == tNULL || len_left == 0 )
                break;  /* no short_bio was entered */

            /* copy blob data block and add the null terminator */
            memcpy(biobuf, val->vt.lvv.buf, val->len);
            biobuf[val->len] = '\0';
            printf("            : %s\n", biobuf);
        }
    }
}
    ...
```

SQL User's Guide

Retrieving Select Statement Results

Retrieving Blob Data Values

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-1	errNOMOREDATA	02000	no more data
-5	errINVHANDLE	02002	invalid connection or statement handle
33	errCOLNUMBER	07009	invalid descriptor index (column number)
52	errCURSTATE	24000	invalid cursor state
53	errNOTSELECT	24000	current statement is not SELECT
141	errBLOBEXPR	42000	blobs cannot be referenced in expressions in deferred mode

rsqlGetDateFormat

Get the current date constant format setting for the connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetDateFormat(
    HCONN              hConn,
    int16_t           *pDatefmt)
```

Arguments

`hConn`	`(input)`	The connection handle.
`pDatefmt`	`(output)`	Pointer to variable into which the current date constant format code will be returned. Possible values are: 1=MMDDYYYY, 2=YYYYMMDD, 3=DDMMYYYY.

Description

Call `rsqlGetDataFormat` to retrieve the format code that indicates the order in which the month, day, and year values are interpreted by SQL in data constants for the connection specified by `hConn`. The format code is returned in argument `*pDatefmt`. The following constants have been defined in the standard RDMe SQL header files for use with this function.

```
#define MMDDYYYY 1
#define YYYYMMDD 2
#define DDMMYYYY 3
```

Of course the "DD" represents the day, "MM" the month, and "YYYY" the year. Thus, July 22, 2011 would be represented as follows for each of the three possible formats:

```
Format 1: date "07-22-2011"
Format 2: date "2011-07-22"
Format 3: date "22-07-2011"
```

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
int16_t    fmt;
```

```
char       sep;
       ...
       /* Display the current date constant format */
       rsqlGetDateFormat(hdbc, &fmt);
       rsqlGetDateSeparator(hdbc, &sep);
       printf("current date constant format is: ");
       switch ( fmt ) {
              case MMDDYYYY: printf("date \"mm%cdd%cyyyy\"\n", sep, sep); break;
              case YYYYMMDD: printf("date \"yyyy%cmm%cdddd\"\n", sep, sep); break;
              case DDMMYYYY: printf("date \"mm%cdd%cyyyy\"\n", sep, sep); break;
       }
       ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
135	errINVDATEFMT	42000	invalid date format

See Also

rsqlGetDateSeparator

rsqlSetDateFormat

rsqlSetDateSeparator

rsqlGetDateSeparator

Get the current date constant separator character for the connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetDateSeparator(
    HCONN              hConn,
    char              *pDatesep)
```

Arguments

`hConn`	`(input)`	The connection handle.
`pDatesep`	`(output)`	Pointer to a char variable into which the current date constant separator character will be returned. The default separator character is a dash ('-').

Description

Call `rsqlGetDataSeparator` to retrieve the character symbol used to separate the month, day, and year values in data constants for the connection specified by `hConn`.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
int16_t    fmt;
char       sep;
       ...
       /* Display the current date constant format */
       rsqlGetDateFormat(hdbc, &fmt);
       rsqlGetDateSeparator(hdbc, &sep);
       printf("current date constant format is: ");
       switch ( fmt ) {
               case MMDDYYYY: printf("date \"mm%cdd%cyyyy\"\n", sep, sep); break;
               case YYYYMMDD: printf("date \"yyyy%cmm%cdddd\"\n", sep, sep); break;
               case DDMMYYYY: printf("date \"mm%cdd%cyyyy\"\n", sep, sep); break;
       }
       ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlGetDateFormat

rsqlSetDateFormat

rsqlSetDateSeparator

rsqlGetDBNames

Get a list of the names of the currently opened databases

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetDBNames(
    const HCONN   hConn,
    const char   *dbnames,
    uint16_t      lenDbnames,
    uint16_t     *pLenDbnames)
```

Arguments

`hConn`	`(input)`	Connection handle.
`dbnames`	`(output)`	A pointer to a char array into which the names of the currently opened databases will be returned (can be NULL).
`lenDbnames`	`(input)`	The maximum size of the `dbnames` char array (including null terminator).
`pLenDbnames`	`(output)`	The actual size of the list (including null terminator).

Description

This function returns a list of the names of the databases that are currently opened on connection handle `hConn`. The list is returned as a string in `dbnames`. The database names are separated by a semi-colon (";"). The `pLenDbnames` argument, if specified, will contain the actual length of the database name string. If the actual size exceeds `lenDbnames` then the string will be truncated and status code `errTRUNCATE` will be returned.

You can call `rsqlGetDBNames` with a `NULL dbnames` argument in order to get the actual length, allocate the buffer, and then call `rsqlGetDBNames` again to retrieve the string.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
RSQL_ERRCODE stat;
char *dbnames;
uint16_t len;
    ...
    rsqlGetDBNames(hdbc, NULL, 0, &len);
    if ( len > 0 ) {
       dbnames = malloc(len);
       rsqlGetDBNames(hdbc, dbnames, len, NULL);
```

```
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value
29	errINVNULL	HY009	invalid use of null pointer

rsqlGetDBTask

Get the RDMe task handle associated with a connection handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetDBTask(
    HCONN       hConn,
    DB_TASK   **pTask)
```

Arguments

`hConn`	`(input)`	Connection handle.
`pTask`	`(output)`	A pointer to a `DB_TASK` pointer variable.

Description

Function `rsqlGetDBTask` can be used to return the RDMe runtime API task handle associated with connection on `hConn`. This function is only to be used when it is necessary for the application to use the RDMe core-level API.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
DB_TASK *task;
    ...
    rsqlGetDBTask(hdbc, &task);
    ... /* make RDMe runtime API calls */
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlAllocConn

rsqlGetDeferBlobMode

Get a statement's deferred reading mode setting for blob data

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetDeferBlobMode(
    HSTMT      hStmt,
    int16_t   *pMode)
```

Arguments

hStmt	(input)	The statement handle.
pMode	(output)	Pointer to variable into which is returned either a 1 (= reading of blob data is deferred) or 0 (= reading of blob data is not deferred).

Description

This function can be called to return the current deferred blob reading mode setting for the specified statement. A blob column is a column of type **long varchar**, **long wvarchar** or **long varbinary**. When deferred reading is enabled the only way to retrieve the result value of a blob column returned in the result set from a `rsqlFetch` call on a **select** statement is through calls to `rsqlGetData`. This allows the blob data to be retrieved a block at a time. When deferred reading is disabled, the entire blob data value is read into an allocated buffer of type `tCHAR` (for **long varchar** columns), `tWCHAR` (for **long wvarchar** columns), or `tBINARY` (for **long varbinary** columns) that is returned in the `RSQL_VALUE` array returned from the call to `rsqlFetch`.

By default, deferred blob reading mode is disabled. Deferred reading of blobs (and every result column, for that matter) is automatically assumed when `rsqlFetch` is called with a `NULL` result argument.

Example

```
#include "rdmsql.h"
...
RSQL_VALUE *row;
uint32_t   len;
char       blobtxt[66];
HSTMT      hstmt;
int16_t    deferred_blobs;
       ...
       /* enable deferred reading of blobs */
       rsqlGetDeferBlobMode(hstmt, &deferred_blobs);
       rsqlExecDirect(hstmt, "select last_name, short_bio from author");
       while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
               printf("author: %s\n", row[0].vt.cv);
```

```
                if ( deferred_blobs ) {
                        /* retrieve blob value in blocks of up to 65 bytes at a time */
                        while ( rsqlGetData(hstmt, 2, &row[1], 65, &len) == errSUCCESS ) {
                                /* call app function to format blob text line */
                                FormatBlob(&row[1], len, blobtxt);
                        }
                }
                else {
                        FormatBlob(&row[1], 65, blobtxt);
                }
                printf("      : %s\n", blobtxt);
        }
        ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value

See Also

rsqlFetch

rsqlGetData

rsqlSetDeferBlobMode

rsqlGetErrorInfo

Get the message associated with the current error code

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetErrorInfo(
       HRSQL       hndl,
       char       *msgbuf,
       int16_t     lenMsgbuf)
```

Arguments

`hndl`	`(input)`	Either a connection handle or a statement handle.
`msgbuf`	`(output)`	A pointer to a char buffer to contain the error message.
`lenMsgbuf`	`(input)`	The length of `msgbuf` (including null terminator).

Description

Call `rsqlGetErrorInfo` with a statement handle to retrieve an error message associated with the last error code returned by any API function for that particular statement handle. Call `rsqlGetErrorInfo` with a connection handle to retrieve an error message associated with the last error code returned by any API function for any handle associated with the connection handle. Using the connection handle when the last error was on a statement handle will only return a generic description of the error code. A more specific description can be retrieved by using the statement handle.

Any errors returned by `rsqlGetErrorInfo` itself or when `errINVHANDLE` is returned will not change the recorded error state.

Example

```
#include "rdmsql.h"
    ...
HSTMT        hstmt;
RSQL_ERRCODE stat;

#define MSGLEN 133
char   *errid, *errmsg;
char    errinfo[MSGLEN];
    ...
    stat = rsqlExecute(hstmt);
    if ( stat != errSUCCESS ) {
       rsqlGetErrorMsg(stat, &errmsg);
       rsqlGetErrorInfo(hstmt, errinfo, MSGLEN);
```

```
        printf("*** error: %s\n", errid, errmsg, errinfo);
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error code returned from the most recent RDMe SQL API call

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
1	errTRUNCATE	01004	data truncation
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlGetErrorMsg

rsqlGetErrorMsg

Return the id and message associated with a specific error/status code

Prototype

```
const char *EXTERNAL_FCN rsqlGetErrorMsg(
    RSQL_ERRCODE       stat,
    const char       **pErrMsg)
```

```
const char *EXTERNAL_FCN rsqlGetErrorMsgEx(
    RSQL_ERRCODE       stat,
    const char       **pSqlstate,
    const char       **pErrMsg)
```

Arguments

`stat`	`(input)`	The error/status code to be looked up.
`pErrmsg`	`(output)`	A pointer to the char array that will contain the error message.
`pSqlstate`	`(output)`	A pointer to the char array that will contain the SQLSTATE message for the error/status code.

Description

This function can be called to return the message text associated with the specified RDMe SQL error/status code, `stat`.

The return value of the function is a pointer to a string representation of the code itself. Error codes are of type `RSQL_ERRCODE` which is a C enum. It is a string representation of the enum value that is returned. For example, if `stat` is `errNOTOPEN`, the function will return the string "`errNOTOPEN`".

If `stat` is an invalid code then the function will return string "error code out of range".

Example

```
#include "rdmsql.h"
...
RSQL_ERRCODE stat;
const char *errid;
char       *msgbuf;
       ...
       errid = rsqlGetErrorMsg(stat, &msgbuf);
```

```
    printf("%s: %s\n", errid, msgbuf);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

rsqlGetGenCFiles

Get the connection handle's "generate C files" mode

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetGenCFiles(
    const HCONN    hConn,
    int16_t       *pGenCFiles)
```

Arguments

`hConn`	`(input)`	Connection handle.
`pGenCFiles`	`(input)`	A pointer to an `int16_t` variable to contain the generate C files mode: 1=enable, 0=disable.

Description

This function retrieve the value that indicates whether the feature of RDMe SQL to generate the C Language source files when compiling a SQL database schema is enabled. When the feature is enabled, the SQL schema compilation will generate a set of C Language source files associated with the database defined in the schema. The generated source files will be placed in the current working directory of the process that compiled the SQL schema.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
int16_t gencfile_flag;
    ...
    /* turn off generate C file mode if enabled */
    rsqlGetGenCFiles(&gencfile_flag, hdbc);
    if ( gencfile_flag )
        rsqlSetGenCFiles(hdbc, 0);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlSetGenCFiles

rsqlGetNumResultCols

Return the number of result columns

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetNumResultCols(
    HSTMT              hStmt,
    uint16_t          *pNoCols)
```

Arguments

hStmt	(input)	The statement handle.
pNoCols	(output)	A pointer to the variable into which the number of result columns will be returned.

Description

Function `rsqlGetNumResultCols` will return in `*pNoCols` the number of result columns for the **select** statement associated with statement handle `hStmt`. The **select** statement must have been compiled through a prior call to `rsqlPrepare` or `rsqlExecDirect`. The number of result columns is also returned from a call to `rsqlFetch`.

Example

```
#include "rdmsql.h"
...
RSQL_ERRCODE  stat;
RSQL_VALUE   *result_cols;
HSTMT         select;
uint16_t      numcols, fetchcols;
       ...
       stat = rsqlExecDirect(select, "select * from author");
       if ( stat == errSUCCESS ) {
               rsqlGetNumResultCols(select, &numcols);
               while ( rsqlFetch(select, &result_cols, &fetchcols) == errSUCCESS ) {
                       if ( numcols != fetchcols ) {
                               /* this should never happen! */
                               ...
                       }
               }
       }
       ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlFetch

rsqlGetNumParams

Return the number of parameter markers specified in a statement.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetNumParams(
    HSTMT              hStmt,
    uint16_t          *pNopars)
```

Arguments

`hStmt`	`(input)`	Statement handle.
`pNopars`	`(output)`	Pointer to variable to contain the number of parameter markers.

Description

This function can be called to return the number of parameter markers that were specified in the previously compiled (through a prior call to either `rsqlPrepare` or `rsqlExecDirect`) SQL statement associated with handle `hStmt`. The number of parameters will be returned in `*pNopars`.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
uint16_t numpars;
RSQL_ERRCODE stat;
    ...
    stat = rsqlGetNumParams(hstmt, &numpars);
    if ( stat == errSUCCESS )
        printf("number of parameter markers = %u\n", numpars);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlPrepare

rsqlGetParamDescr

Return a description of a parameter marker

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetParamDescr(
    HSTMT              hStmt,
    int16_t            parno,
    SQL_T             *pType,
    uint32_t          *pVarlen,
    int16_t           *pNullable)
```

Arguments

hStmt	(input)	The statement handle.
parno	(input)	The number of the parameter to be described (1..num of parameters).
pType	(output)	Pointer to variable into which is returned the data type of the parameter. NULL is okay.
pVarlen	(output)	Pointer to variable into which is returned the maximum length (in bytes) of parameter of non-scalar data types. NULL is okay.
pNullable	(output)	Pointer to variable into which is returned either a 1 if the parameter can be set to NULL or 0 if not. NULL is okay.

Description

Call function `rsqlGetParamDescr` to retrieve information about a particular SQL statement parameter when it is important to know the data type and maximum length of the table column that is associated with the specified parameter. Note that the `pVarlen` value is only set for parameters associated with non-scalar data type columns (a non-scalar is a type, e.g., **char** or **varchar**, that is always declared with a specified maximum length). The `pNullable` argument will contain a 1 if the column associated with this parameter can be `NULL`. The `pNullable` indicator only applies when the statement associated with `hStmt` is either an **insert** or an **update**.

Note that SQL is not always able to determine the parameter type when the parameter marker is used in the context of a complex expression. When the parameter type cannot be determined, `*pType` will be set to `tNOVAL`.

This function can only be called after the SQL statement associated with `hStmt` has been compiled through a prior call to `rsqlPrepare` or `rsqlExecDirect`.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
```

```
RSQL_ERRCODE stat;
uint16_t nopars;
uint16_t parno;
SQL_T ptype;
uint32_t *parMaxLens, maxlen;
int32_t len = -2;   /* data-at-exec parameter */
void *buf;
        ...
        rsqlGetNumParams(hstmt, &nopars);
        parMaxLens= calloc(nopars, sizeof(uint32_t));

        for (parno = 1; parno <= nopars; ++parno) {
                rsqlGetParamDescr(hstmt, parno, &ptype, &parMaxLens[parno-1], NULL);
                if ( IS_BLOB(ptype) ) {
                        /* bind blob parameter as data-at-exec */
                        rsqlBindParam(hstmt, parno, tCHAR, &parMaxLens[parno-1], &len);
                else
                        ... /* bind non-blob, non-data-at-exec parameter */
        }

        stat = rsqlExecute(hstmt);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
34	errPARNUMBER	07009	invalid descriptor index (parameter number)
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlBindParam

rsqlGetReadOnlyTrmode

Get the current read only transaction mode setting

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetReadOnlyTrmode(
    HCONN              hConn,
    int16_t           *pRomode)
```

Arguments

hConn	(input)	The connection handle.
pRomode	(output)	Returns the current mode setting: 0 = manual, 1 = auto.

Description

Call `rsqlGetReadOnlyTrmode` to get the current read only transaction mode setting. When `*pRomode` is 1, SQL processes all **select** statements within a system-generated read only transaction. When `*pRomode` is set to 0, SQL first issues read lock requests for the tables referenced in the **select** statement.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
int16_t romode;
    ...
    rsqlSetReadOnlyTrmode(hdbc, &romode);
    if ( romode == 1 )
       Display("Read only transaction mode is enabled");
    else
       Display("Read only transaction mode is disabled");
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlSetReadOnlyTrmode

["Read Only Transactions" in "Concurrent Database Access" chapter]

rsqlGetRowCount

Return the number of rows processed by the just-executed statement

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetRowCount(
    HSTMT              hStmt,
    uint64_t          *pNoRows)
```

Arguments

hStmt	(input)	Statement handle.
pNoRows	(output)	A pointer to the variable in which the total rows will be returned.

Description

Call `rsqlGetRowCount` to retrieve the total number of rows processed by the executed **insert**, **update** or **delete** statement associated with `hStmt`. Note that the number of rows affected by an **update** or **delete** statement may be more than just the number of rows of the specified table due to the possibility of cascaded updates or deletes that occur.

Example

```
#include "rdmsql.h"
    ...
HSTMT        hstmt;
RSQL_ERRCODE stat;
uint64_t     mgr_count;
    ...
    /* fire all managers */
    stat = rsqlPrepare(hstmt, "delete from salesperson where mgr_id is null",
        NULL, NULL, NULL);
    if ( stat == errSUCCESS ) {
        stat = rsqlExecute(hstmt);
        if ( stat == errSUCCESS ) {
            rsqlGetRowCount(hstmt, &mgr_count);
            printf("%I64u managers were deleted\n", mgr_count);
        }
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state

rsqlGetSelectType

Open a database on connection.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetSelectType(
    HSTMT               hStmt,
    SELECT_TYPE        *pSeltype)
```

Arguments

`hStmt`	`(input)`	Statement handle.
`pSeltype`	`(output)`	A pointer to the variable to contain the **select** type value.

Description

This function can be called to determine the type of **select** statement associated with statement handle `hstmt`.

The statement must have been compiled through a previous call to `rsqlPrepare`. If not, `rsqlGet-SelectType` will return error code `errINVSTATE`. If the compiled statement was not a **select** statement error code `errNOTSELECT` will be returned.

The types of **select** statements are identified as one of the values of the `SELECT_TYPE` enumerator as shown in the table below.

Select Statement Type Identification

Constant	Corresponding SQL Statement
`sqlSELECT`	**select** ...
`sqlINSERT`	**insert into** *tabname* ...
`sqlUPDATE`	**update** tabname ...

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
SELECT_TYPE stype;
    ...
    stat = rsqlPrepare(hstmt, "select * from author for read only");
    if ( stat == errSUCCESS ) {
        rsqlGetSelectType(hstmt, &stype);
        if ( stype != sqlREADONLY )
            ... something is awfully, awfully wrong!
    ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
53	errNOTSELECT	24000	current statement is not SELECT
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlGetStmtType

rsqlGetStmtType

Get the statement type.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetStmtType(
    HSTMT               hStmt,
    STMT_TYPE          *pStype)
```

Arguments

`hStmt`	`(input)`	Statement handle.
`pStype`	`(output)`	A pointer to the variable to contain the statement type value.

Description

This function can be called to determine the type of SQL statement associated with statement handle `hstmt`. The statement must have been compiled through a previous call to `rsqlPrepare`. If not, `rsqlGetStmtType` will return error code `errINVSTATE`.

Statement types are identified as one of the values of the `STMT_TYPE` enumerator as shown in the table below.

Table 26. Statement Type Identification Constants of Type STMT_TYPE

Constant	Corresponding SQL Statement
`sqlSELECT`	**select** ...
`sqlINSERT`	**insert into** *tabname* ...
`sqlUPDATE`	**update** tabname ...
`sqlOPEN`	**open** *dbname*
`sqlDELETE`	**delete from** *tabname* ...
`sqlSTART`	**start transaction** ...
`sqlSAVEPOINT`	**savepoint** ...
`sqlRELEASE`	**release savepoint** ...
`sqlCOMMIT`	**commit transaction**
`sqlROLLBACK`	**rollback transaction**
`sqlCRPROC`	**create procedure** ...
`sqlDRPROC`	**drop procedure** ...
`sqlEXECUTE`	**execute** *procname* ...
`sqlSET`	**set** {**auto commit** | **debug** } ...
`sqlSETCOLUMN`	**set column stats** ...
`sqlLOCK`	**lock table** ...
`sqlUNLOCK`	**unlock table** ...
`sqlIMPORT`	**import** ...

`sqlEXPORT`	**export** ...
`sqlDDL`	SQL DDL statement

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
STMT_TYPE stype;
    ...
    stat = rsqlPrepare(hstmt, "select * from author");
    if ( stat == errSUCCESS ) {
       rsqlGetStmtType(hstmt, &stype);
       if ( stype != sqlSELECT )
           ... something is awfully, awfully wrong!
    ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlGetSelectType

rsqlGetTableName

Get the name of the table in which the **select** result column is defined

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlGetTableName(
    HSTMT           hStmt,
    int16_t         colno,
    char           *dbname,
    uint16_t        lenDbname,
    uint16_t       *pLenDbname,
    char           *tabname,
    uint16_t        lenTabname,
    uint16_t       *pLenTabname)
```

Arguments

hstmt	(input)	Statement handle.
colno	(input)	The **select** statement result column number (1..# of columns).
dbname	(output)	A pointer to the char buffer in which the database name will be returned (can be NULL).
lenDbname	(input)	The length of the dbname buffer (including null terminator).
pLenDbname	(output)	The length of the resulting database name (including null terminator).
tabname	(output)	A pointer to the char buffer in which the table name will be returned (can be NULL).
lenTabname	(input)	The length of the tabname buffer (including null terminator).
pLenTabname	(output)	The length of the resulting table name (including null terminator).

Description

Call rsqlGetTableName to get the name of the database and table in which the specified **select** statement result column is declared. The ordinal column number is given by the colno argument where the columns are numbered from left to right beginning with 1. If the specified column references a single table column and not an expression, the name of its database is returned in dbname and the name of the table is returned in tabname otherwise tabname will contain a null string (i.e., "\0").

You can call rsqlGetTableName with a NULL dbname/tabname argument to discover the length of the database/table name string (returned in pLenDbname/pLenTabname) if you want to dynamically allocate for the dbname/tabname buffer to be used in a second rsqlGetTableName call to retrieve the database and table names. However, the maximum length (including null byte) of a table (or column or database) name is given by the NAMELEN #define that is available through the RDMe SQL standard header files.

Example

```
#include "rdmsql.h"
    ...
    HSTMT        hstmt;
    RSQL_ERRCODE stat;
    int16_t colNo;
    stat = rsqlPrepare(hstmt, "select * from patron");
    for (colNo = 1; stat == errSUCCESS; colNo++) {
        char  dbname[NAMELEN];
        char  tabname[NAMELEN];
        stat = rsqlGetTableName(hstmt, colNo, dbname, NAMELEN, NULL,
                                tabname, NAMELEN, NULL);
        if (stat == errSUCCESS) {
            /* We now have the tablename and the database name for one column */
        }
    }
    if (stat == errCOLNUMBER) {
        /* We got all columns */
        stat = errSUCCESS;
    }
    else {
       /* Some error occured */
    }
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
33	errCOLNUMBER	07009	invalid descriptor index (column number)
53	errNOTSELECT	24000	current statement is not SELECT
65	errINVSTATE	RX008	invalid statement state

See Also

rsqlAllocStmt

rsqlInitDB

Initialize a database

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlInitDB(
    HSTMT    hConn,
    char    *dbname)
```

Arguments

hConn	(input)	The connection handle.
dbname	(input)	Pointer to string that contains the name of the database to be initialized.

Description

Function `rsqlInitDB` can be called to initialize the contents of database `dbname`. The specified database must have been opened in exclusive access mode. Also, a transaction must not be active when this function is called. This function will re-initialize the database irrecoverably destroying the entire contents of the database.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
HSTMT hstmt;
       ...
       /* re-initialize and reload the bookshop database */
       rsqlAllocConn(&hdbc, 0);
       stat = rsqlOpenDB(hdbc, "bookshop", "x");
       if ( stat != errSUCCESS ) {
               ... /* trouble opening db */
               return stat;
       }
       rsqlAllocStmt(hdbc, &hstmt);
       rsqlInitDB(hdbc, "bookshop");
       rsqlTransStart(hdbc, "bookshop_import");
       rsqlExecDirect(hstmt, "import into acctmgr from file \"acctmgrs.txt\"");
       ...
       rsqlExecDirect(hstmt, "import into sale from file \"sales.txt\"");
       rsqlTransCommit(hdbc);
       ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
36	errTRACTIVE	25000	transaction is active
41	errDBNOTOPEN	42000	database not open
138	errEXCLUSIVE	RX020	operation requires exclusive database access

rsqlLockTables

Issue an explicit lock request for one or more database tables

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlLockTables(
    HCONN           hConn,
    const char     *dbname,
    int16_t         noTablocks,
    TABLE_LOCK     *tablocks)
```

Arguments

hdbc	(input)	The connection handle.
dbname	(input)	Pointer to string containing the name of the database containing the tables to be locked.
noTablocks	(input)	The number of tables in the tablocks array.
tablocks	(input)	Array of TABLE_LOCK entries specifying the tables to be locked and the type of lock to be applied.

Description

You can call rsqlTableLocks to explicitly request locks on one or more tables in a given database. Normally, SQL automatically issues any needed lock requests in the processing of each **select**, **insert**, **update**, and **delete** statement and in the execution of a stored procedure. However, if you believe it is important to explicitly lock specific tables, then you may use this function to do so.

When you have explicitly issued table locks through either a call to this function or execution of the **lock table** SQL statement you disable SQL's automatic lock mode. This means that the execution of any SQL statement that does not have all of the reference tables locked will return an error (errNOTLOCKED) indicating that the table must first be explicitly locked. Note that the values of foreign key columns are retrieved from the referenced row in the primary key table (RDMe SQL does not actually store them in the foreign key table). Hence, *both* the foreign and primary key tables must be explicitly locked when accessing foreign key column values.

If any implicit locks are active on the connection then explicit locking is not allowed and rsqlLockTables will return error code errLOCKMODE.

Error code errROTRACT will be returned when a read-only transaction is active. Read-only transactions only allow database reading and locks are not required.

Error code errAUTOCOMMIT will be returned if auto commit mode is enabled as explicit locking is not allowed in auto commit mode.

Individual table locks are specified through the tablocks array. The TABLE_LOCK structure and declaration is given below along with the LOCK_TYPE declaration.

```
typedef enum _lock_type {
    lockDEFAULT = 0,                /* not specified */
    lockREAD = 'r',                 /* read lock */
    lockWRITE = 'w'                 /* write lock */
} LOCK_TYPE;

/* table lock request packet definition */
typedef struct table_lock {
    char           name[NAMELEN];   /* name of the table to be locked */
    LOCK_TYPE      type;            /* kind of lock to apply: read, write, default
*/
} TABLE_LOCK;
```

For each `tablocks` array entry you need to specify the name of the table and the lock type. The `lock-DEFAULT` type will be interpreted as a write lock if a transaction is active and as a read lock if a transaction is not active. An attempt to request a write lock when a transaction is not active will result in an error.

Either all of the locks will be granted or none will be. Status code `errTIMEOUT` is returned in the event that one or more of the locks are not granted (due to some other connection already having the table locked) within the timeout period associated with the connection.

Read locks that were granted prior to a transaction or read and write locks that were granted within a transaction are freed when the transaction ends, through execution of a transaction commit or rollback.

Read locks granted outside of a transaction must be freed either through a call to `rsqlUnlockTable`, to free a read lock on a specific table, or through a call to `rsqlUnlockTableAll` which frees all table read locks, or through an explicit call to `rsqlTransCommit`.

Executing one or more select statements without explicitly requesting locks will implicitly acquire the necessary locks. These read locks will be free when all the result rows have been fetched, the statement handles have been closed or freed, or with an explicit commit.

An insert, delete, or searched update will acquire the needed write locks. Read locks held prior to execution of these statements will instead be upgraded to write locks and when the execution is completed downgrade back to read locks.

A positioned update will temporarily upgrade the read lock to a write lock and when the update is completed downgrade it back to a read lock.

Example

```
#include "rdmsql.h"
    ...
RSQL_ERRCODE stat;
void *hdbc;
TABLE_LOCK lock_request[] = {
       {"author", lockREAD},
       {"book", lockREAD}
};
```

```
int16_t notabs = sizeof(lock_request)/sizeof(TABLE_LOCK);
    ...
    stat = rsqlLockTables(hdbc, "bookshop", notabs, lock_request);
    if ( stat == errSUCCESS ) {
        ... /* issue and process SELECT statement on author and book tables */
        rsqlUnlockTableAll(hdbc);
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
14	errTABNOTFOUND	42S02	table not found
41	errDBNOTOPEN	42000	database not open
28	errINVARG	HY009	invalid argument value
118	errTIMEOUT	HYT00	timeout expired
129	errROTRACT	25000	read-only transaction is active
162	errLOCKMODE	RX026	illegal locking mode
164	errAUTOCOMMIT	42000	operation not allowed when autocommit is enabled

SQL User's Guide

Locking in RDMe SQL

See Also

rsqlSetTimeout

rsqlUnlockTable

rsqlUnlockTableAll

rsqlMoreResults

Execute the next statement in the currently executing stored procedure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlMoreResults(
    HSTMT             hStmt)
```

Arguments

`hStmt`	`(input)`	Statement handle.

Description

Function `rsqlMoreResults` is called to execute the next **select** statement contained in a retrieval stored procedure. If there are no more statements left to be executed or if the original statement was not an **execute** statement, then this function returns status `errNOMOREDATA`. In all other respects, this function is equivalent to `rsqlExecute`..

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
    ...
    stat = rsqlPrepare(hstmt, "execute myproc");
    if ( stat == errSUCCESS ) {
       stat = rsqlExecute(hstmt);
       while ( stat == errSUCCESS ) {
               ... call rsqlFetch to process each result set
               stat = rsqlMoreResults(hstmt);
       }
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-1	errNOMOREDATA	02000	no more data
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error

See Also

rsqlExecute

rsqlOpenCat

Open a database via its compiled database catalog

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlOpenCat(
       HCONN          *hdbc,
       const SYSDB    *pCat,
       const char     *dbspec,
       const char     *mode)
```

Arguments

hConn	(input)	Connection handle.
pCat	(input)	A pointer to the catalog structure variable for the database.
dbspec	(input)	A string specifying the database name and/or TFS specification or database union specification.
mode	(input)	A string specifying the database open mode: "s" = shared, "x" = exclusive.

Description

Call `rsqlOpenCat` to open a database through use of its compiled catalog module. You can separately call `rsqlOpenCat` for the catalog of each database to be used within connection `hConn`. The `pCat` argument is the address of the `SYSDB` variable associated with a database catalog module that has been compiled and linked with your application program. This function can only be called after having called `rsqlAllocConn` to allocate and open connection `hConn`.

The `dbspec` argument, if specified, contains the location specification for the TFS on which the database resides and, optionally, the name of the database if different than the one that was identified by the create database statement associated with the specified catalog, `pCat`. The `dbspec` string must be constructed as shown below.

[dbname]@TFSComputerName[:port]

The default TCP/IP listening port number is 21553. If `dbspec` is NULL then the database to be opened will have the name specified in the create database statement and be assumed to reside on the document root directory on the default TFS.

You can also use `dbspec` to specify a union of two or more databases by separating each database entry with the "|" symbol as shown in the example below.

```
nsfdb1@NSFServer1:21553|nsfdb2@NSFServer2:21555|nsfdb3@NSFServer3:21557
```

In a database union, the individual (partitioned) databases running on the separate TFSs are treated by the system as a single database. If a database union is specified then no non-union databases can be opened in the same connection and no other database opens are allowed.

Once a database has been opened, subsequent calls to `rsqlOpenCat` (or `rsqlOpenDB`) must specify the same database open mode as the first.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
#include "bookshop_cat.h"
#include "nsfawards_cat.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
        ...
stat = rsqlAllocConn(&hdbc);
if ( stat == errSUCCESS ) {
        stat = rsqlOpenCat(hdbc, &bookshop_cat, "@RaimaServer1:21553", "s");
        if ( stat == errSUCCESS )
            stat = rsqlOpenCat(hdbc, &nsfawards_cat, "@RaimaServer2:21555", "s");
        if ( stat != errSUCCESS ) {
            printf("unable to open the databases\n");
            ...
...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
23	errNODB	42000	unable to open database
29	errINVNULL	HY009	invalid use of null pointer
36	errTRACTIVE	25000	transaction is active

39	errCONNECTED	08000	must call before connect
137	errINVOPENMODE	42000	invalid db open mode
169	errDBOPENMODE	42000	another database is already open in different mode
175	errUNIONOPEN	42000	db union open invalid when other database is open
176	errDBUNAVAIL	42000	database unavailable due to exclusive access rules
177	errDBOPEN	42000	database has already been opened

See Also

rsqlOpenDB

rsqlCloseDB

rsqlCloseDBAll

rsqlOpenDB

Open one or more databases

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlOpenDB(
    HCONN               hConn,
    const char         *dbname,
    const char         *mode)
```

Arguments

hConn	(input)	Connection handle.
dbname	(input)	A string specifying the database names/TFS specifications or database union specification to be opened.
mode	(input)	A string specifying the database open mode: "s" = shared, "x" = exclusive.

Description

Call `rsqlOpenDB` to open the databases specified in the `dbnames` argument string as described below. This function can only be called after having called `rsqlAllocConn` to allocate and open connection `hConn`.

The `dbnames` argument contains the name of the database and, optionally, the location specification for the TFS on which the database resides. If more than one database is to be opened then each database name must be separated by a semicolon, ";". The dbnames string must be constructed as shown below.

dbname[*@TFSComputerName*[*:port*]][*;dbname*[*@TFSComputerName*[*:port*]]...

If *TFSComputerName* is specified with the ":*port*" number, port is assumed to be 21553. If no TFS is specified (i.e., *dbname* only) then the database to be opened will be assumed to reside on the document root directory on the default TFS.

You can also use `dbnames` to specify a union of two or more databases by separating each database entry with the "|" symbol as shown in the example below.

```
nsfdb1@NSFServer1:21553|nsfdb2@NSFServer2:21555|nsfdb3@NSFServer3:21557
```

In a database union, the individual (partitioned) databases running on the separate TFSs are treated by the system as a single database. If a database union is specified then no non-union databases can be opened in the same connection and no other database opens are allowed.

Once a database has been opened, subsequent calls to `rsqlOpenDB` (or `rsqlOpenCat`) must specify the same database open mode as the first.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
        ...
stat = rsqlAllocConn(&hdbc);
if ( stat == errSUCCESS ) {
        stat = rsqlOpenDB(hdbc, "bookshop@Srvr1:21553;nsfawards@Srvr2:21555", "s");
        if ( stat != errSUCCESS ) {
                printf("unable to open the databases\n");
                ...
        }
        ... access the databases
...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
23	errNODB	42000	unable to open database
29	errINVNULL	HY009	invalid use of null pointer
36	errTRACTIVE	25000	transaction is active
39	errCONNECTED	08000	must call before connect
137	errINVOPENMODE	42000	invalid db open mode
169	errDBOPENMODE	42000	another database is already open in different mode
175	errUNIONOPEN	42000	db union open invalid when other database is open
176	errDBUNAVAIL	42000	database unavailable due to exclusive access rules

177	errDBOPEN	42000	database has already been opened

See Also

rsqlOpenCat

rsqlCloseDB

rsqlCloseDBAll

rsqlPackDate

Pack a CAL_DATE into a binary DATE_VAL

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlPackDate(
    const CAL_DATE *pUDt,
    DATE_VAL       *pPDt)
```

Arguments

pUDt	(input)	Pointer to unpacked date value.
pPDt	(output)	Pointer to variable to contained packed date value.

Description

Call `rsqlPackDate` to convert an unpacked date value contained in the `CAL_DATE struct` variable pointed to by `pUDt` into the packed date value that will be returned in the `DATE_VAL` variable pointed to by `pPDt`. The `CAL_DATE struct` and `DATE_VAL typedef` are given below.

```
typedef uint32_t DATE_VAL;

typedef struct tagCAL_DATE {
    int32_t  year;      /* 1 - ???? */
    uint16_t month;     /* 1 - 12 */
    uint16_t day;       /* 1 - 31 */
} CAL_DATE;
```

This function is needed to convert date values into the packed `tDATE` format when passing `RSQL_VALUE` date arguments to stored procedures via calls to `rsqlExecProc`.

The error messages associated with the error codes returned by this function can only be retrieved by calling `rsqlGetErrorMsg` as `rsqlGetErrorInfo` only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

Stored procedure declaration:

```
create proc GetBookByDate(acquired date) as
       select * from book where date_acqd = acquired
end proc;
```

C program snippet:

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE val, *rows;
uint16_t  nocols;

CAL_DATE  cdate = {2007, 5, 8};
       ...
       /* retrieve books acquired on May 8th, 2007 */
       rsqlPackDate(&cdate, &val.vt.dtv);
       val.type = tDATE;
       stat = rsqlExecProc(hstmt, "getbookbydate", 1, &val);
       if ( stat == errSUCCESS ) {
               while ( rsqlFetch(hstmt, &rows, &nocols) == errSUCCESS ) {
                       ... fetch result rows
       ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
15	errDATETIMEOVF	22008	date/time value overflow
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlUnpackDate

rsqlPackTime

rsqlPackTime

Pack a CAL_TIME into a binary TIME_VAL

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlPackTime(
    const CAL_TIME *pUTm,
    TIME_VAL       *pPTm)
```

Arguments

pUTm	(input)	Pointer to unpacked time value.
pPTm	(output)	Pointer to variable to contained packed time value.

Description

Call `rsqlPackTime` to convert an unpacked time value contained in the `CAL_TIME struct` variable pointed to by `pUTm` into the packed time value that will be returned in the `TIME_VAL` variable pointed to by `pPTm`. The `CAL_TIME struct` and `TIME_VAL typedefs` are given below.

```
typedef uint32_t TIME_VAL;

typedef struct tagCAL_TIME {
    uint16_t hour;      /* 0 - 23 */
    uint16_t minute;    /* 0 - 59 */
    uint16_t second;    /* 0 - 59 */
    uint16_t fraction;  /* 0 - 9999 */
} CAL_TIME;
```

This function is needed to convert time values into the packed `tTIME` format when passing `RSQL_VALUE` time arguments to stored procedures via calls to `rsqlExecProc`.

The error messages associated with the error codes returned by this function can only be retrieved by calling `rsqlGetErrorMsg` as `rsqlGetErrorInfo` only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

Stored procedure declaration:

```
create proc GetActiveAuctions(btm time) as
       select * from auction where current_date() between start_date and end_date
              and hour(btm) = hour(bid_time)
end proc;
```

C program snippet:

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE val, *rows;
uint16_t  nocols;

CAL_TIME  cdtime = {16, 0, 0, 0};
       ...
       /* retrieve list of auctions active during 8 PM hour */
       rsqlPackTime(&cdtime, &val.vt.tmv);
       val.type = tTIME;
       stat = rsqlExecProc(hstmt, "GetActiveAuctions", 1, &val);
       if ( stat == errSUCCESS ) {
              while ( rsqlFetch(hstmt, &rows, &nocols) == errSUCCESS ) {
                     ... fetch result rows
       ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
15	errDATETIMEOVF	22008	date/time value overflow
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlUnpackTime

rsqlPackTimestamp

Pack a CAL_TIMESTAMP into a binary TIMESTAMP_VAL

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlPackTimestamp(
    const CAL_TIMESTAMP *pUTs,
    TIMESTAMP_VAL       *pPTs)
```

Arguments

pUTs	(input)	Pointer to unpacked timestamp value.
pPTs	(output)	Pointer to variable to contained packed timestamp value.

Description

Call `rsqlPackTimestamp` to convert an unpacked timestamp value contained in the `CAL_TIMESTAMP struct` variable pointed to by `pUTs` into the packed time value that will be returned in the `TIMESTAMP_VAL` variable pointed to by `pPTs`. The `CAL_TIMESTAMP struct` and `TIMESTAMP_VAL typedefs` are given below.

```
typedef struct {
    DATE_VAL date;
    TIME_VAL time;
} TIMESTAMP_VAL;

typedef struct tagCAL_TIMESTAMP {
    int32_t  year;      /* 1 - ???? */
    uint16_t month;     /* 1 - 12 */
    uint16_t day;       /* 1 - 31 */
    uint16_t hour;      /* 0 - 23 */
    uint16_t minute;    /* 0 - 59 */
    uint16_t second;    /* 0 - 59 */
    uint16_t fraction;  /* 0 - 9999 */
} CAL_TIMESTAMP;
```

This function is needed to convert timestamp values into the packed `tTIMESTAMP` format when passing `RSQL_VALUE` timestamp arguments to stored procedures via calls to `rsqlExecProc`.

The error messages associated with the error codes returned by this function can only be retrieved by calling `rsqlGetErrorMsg` as `rsqlGetErrorInfo` only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

Stored procedure declaration:

```
create proc GetBidsByDate(bts timestamp) as
       select * from auction where convert(bid_ts, tDATE) = convert(bts, tDATE)
end proc;
```

C program snippet:

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE val, *rows;
uint16_t  nocols;

CAL_TIMESTAMP cts;
       ...
       /* retrieve list of bids made on user-spec'd date */
       GetDateFromUser(&cts);
       rsqlPackTimestamp(&cts, &val.vt.tsv);
       val.type = tTIMESTAMP;
       stat = rsqlExecProc(hstmt, "GetBidsByDate", 1, &val);
       if ( stat == errSUCCESS ) {
               while ( rsqlFetch(hstmt, &rows, &nocols) == errSUCCESS ) {
                       ... fetch result rows
       ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
15	errDATETIMEOVF	22008	date/time value overflow

29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlUnpackTimestamp

rsqlPackDate

rsqlPackTime

rsqlParamData

Check for and initialize `rsqlPutData` for next data-at-exec parameter

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlParamData(
    HSTMT              hStmt,
    uint16_t          *pParno,
    void             **pValue)
```

Arguments

`hStmt`	`(input)`	The statement handle.
`parno`	`(output)`	Pointer to variable into which the number of the next parameter will be returned. NULL is okay.
`pValue`	`(output)`	Pointer to the parameter value pointer. NULL is okay.

Description

Function `rsqlParamData` must be called whenever one or more data-at-exec parameters have been bound in a given statement. The call to `rsqlExecute` will return status code `errNEEDDATA` to indicate that the statement includes data-at-exec parameters. You then need to call `rsqlParamData` to set up the subsequent calls to rsqlPutData to supply the data values for each data-at-exec parameter that has been bound to the statement. Status code `errNEEDDATA` is returned from `rsqlParamData` as long as there is another data-at-exec parameter that must be supplied a data value. After all parameters have been processed `rsqlParamData` will return status `errSUCCESS` indicating that execution of the statement has completed.

In RDMe SQL, only blob data fields (**long varchar**, **long wvarchar**, and **long varbinary**) qualify as data-at-exec parameters. Moreover, only insert and update statements that involve changes to one or more blob table columns will require a call to `rsqlParamData`.

The number of the next data-at-exec parameter (1..# of parameters) will be returned in `*pParno`, if specified. The value pointer that was originally specified in the associated `rsqlBindParam` call will be returned in `*pValue`, if specified.

Example

```
#include "rdmsql.h"
    ...
HSTMT hstmt;
RSQL_ERRCODE stat;
uint16_t     nopars;
uint16_t    parno;
```

```
SQL_T        ptype;
uint32_t  *parMaxLens, maxlen;
int32_t      len = -2;   /* data-at-exec parameter */
char *buf;
    ...
    rsqlGetNumParams(hstmt, &nopars);
    parMaxLens= calloc(nopars, sizeof(uint32_t));

    for (parno = 1; parno <= nopars; ++parno) {
       rsqlGetParamDescr(hstmt, parno, &ptype, &parMaxLens[parno-1], NULL);
       if (ptype == tCLOB) ) {
           /* bind blob parameter as data-at-exec */
           rsqlBindParam(hstmt, parno, tCHAR, &parMaxLens[parno-1], &len);
       else
           ... /* bind non-blob, non-data-at-exec parameter */
       }

       stat = rsqlExecute(hstmt);

       if ( stat == errNEEDDATA ) {
           while ( rsqlParamData(hstmt, &parno, &maxlen) == errNEEDDATA ) {
               /* call app function that gets next block of blob data */
               while ( GetNextDataBlock(parno, &buf, &len, maxlen) ) {
                   stat = rsqlPutData(hstmt, buf, len);
                   if ( stat != errSUCCESS )
                       ...
               }
           }
           ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-4	errNEEDDATA	01000	need data
-5	errINVHANDLE	02002	invalid connection or statement handle

65	errINVSTATE	RX008	invalid statement state

See Also

rsqlBindParam

rsqlGetParamDescr

rsqlGetNumParams

rsqlPutData

rsqlPrepare

Compile an SQL statement

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlPrepare(
    HSTMT              hstmt,
    const char        *sqlstr)
```

Arguments

hstmt	(input)	Statement handle.
sqlstr	(input)	A pointer to the character string containing the SQL statement to be compiled.

Description

You must call `rsqlPrepare` in order to prepare a RDMe SQL statement for execution (i.e., compile the statement). Status code `errSUCCESS` is returned when the specified statement has compiled without error. Otherwise, an error code that identifies the error will be returned.

> A left (right) join where the right (left) table does not have a key or a foreign key reference to the left (right) table is not currently supported.

Example

```
#include "rdmsql.h"
    ...
HSTMT        hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE  *row;
char         last_name[15] = "B%";    /* authors with last name that begins with
"B" */
    ...
    stat = rsqlPrepare(hstmt, "select * from author where last_name like ?");
    if ( stat == errSUCCESS ) {
        stat = rsqlBindParam(hstmt, 1, tCHAR, last_name, NULL);
        if ( stat == errSUCCESS ) {
            stat = rsqlExecute(hstmt);
            while ( stat == errSUCCESS )
                stat = rsqlFetch(hstmt, &row);
        }
    }
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
18	errNOMEMORY	HY013	memory management error
29	errINVNULL	HY009	invalid use of null pointer
65	errINVSTATE	RX008	invalid statement state
69	errDDLNOTEXEC	42000	prior prepared DDL statement not executed

See Also

rsqlExecute

rsqlExecDirect

rsqlPutData

Put a data value for a data-at-exec blob parameter

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlPutData(
    HSTMT              hStmt,
    const void        *value,
    uint32_t           lenValue)
```

Arguments

hStmt	(input)	The statement handle.
value	(input)	Pointer to parameter data value buffer.
lenValue	(input)	Length in bytes of data value.

Description

Function `rsqlPutData` must be called whenever a data-at-exec parameter has been bound to a blob data column being changed in an **insert** or **update** statement. The call to `rsqlExecute` will return status code `errNEEDDATA` to indicate that the statement includes data-at-exec parameters. You then need to call `rsqlParamData` to set up the subsequent calls to `rsqlPutData` to supply the data values for each data-at-exec parameter that has been bound to the statement. Status code `errNEEDDATA` is returned from `rsqlParamData` as long as there is another data-at-exec parameter that must be supplied a data value. After all parameters have been processed `rsqlParamData` will return status `errSUCCESS` indicating that execution of the statement has completed.

In RDMe SQL, only blob data fields (**long varchar**, **long wvarchar**, and **long varbinary**) qualify as data-at-exec parameters. Moreover, only **insert** and **update** statements that involve changes to more that one blob table column will require calls to `rsqlPutData`.

You can make as many calls to `rsqlPutData` as needed to provide all of the blob data. Each subsequent call, appends `lenValue` bytes from the data buffer pointed to by `value` to the data already specified in prior calls to `rsqlPutData` for the same parameter. When all of the data has been sent, call `rsqlParamData` to indicate that you have completed processing the data for the last parameter and to set up for the next parameter if there is one.

Example

```
#include "rdmsql.h"
    ...
HSTMT        hstmt;RSQL_ERRCODE stat;
uint16_t      nopars;
uint16_t      parno;
```

```
SQL_T         ptype;
uint32_t     *parMaxLens, maxlen;
int32_t       len = -2;   /* data-at-exec parameter */
uint32_t     buflen;
char        *buf;
    ...
    rsqlGetNumParams(hstmt, &nopars);
    parMaxLens= calloc(nopars, sizeof(uint32_t));

    for (parno = 1; parno <= nopars; ++parno) {
        rsqlGetParamDescr(hstmt, parno, &ptype, &parMaxLens[parno-1], NULL);
        if (ptype == tCLOB) {
            /* bind blob parameter as data-at-exec */
            rsqlBindParam(hstmt, parno, tCHAR, &parMaxLens[parno-1], &len);
        else
            ... /* bind non-blob, non-data-at-exec parameter */
        }

        stat = rsqlExecute(hstmt);
        if ( stat == errNEEDDATA ) {
            while ( rsqlParamData(hstmt, &parno, &maxlen) == errNEEDDATA ) {
                /* call app function that gets next block of blob data */
                while ( GetNextDataBlock(parno, &buf, &buflen, maxlen) ) {
                    stat = rsqlPutData(hstmt, buf, buflen);
                    if ( stat != errSUCCESS )
                        ...
                }
            }
            ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
35	errFCNSEQ	HY010	function call sequence error

65	errINVSTATE	RX008	invalid statement state
97	errFCNARG	21000	incorrect number of function arguments

See Also

rsqlBindParam

rsqlParamData

rsqlGetParamDescr

rsqlGetNumParams

rsqlRegisterProc

Register a compiled stored procedure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlRegisterProc(
    HDBC               hConn,
    PROC_EXEC         *pProc)
```

Arguments

`hConn`	`(input)`	Connection handle.
`pProc`	`(input)`	A pointer to the compiled stored procedure packet.

Description

Function `rsqlRegisterProc` must be called to register a pre-compiled stored procedure that has been linked with your application. This provides the RDMe SQL engine with the pointer to the stored procedure packet so that it can be executed when invoked either by a call to function `rsqlExecProc` or execution of the **execute** SQL statement. The packet is made available by `#include`-ing the header file generated by SQL when the stored procedure was compiled. The stored procedure packet variable is named *`procname`*`_defn`.

This function must be called before any attempt is made to execute the associated stored procedure.

Example

```
#include "rdmsql.h"
#include "findbook_ssp.h"
    ...
HCONN hdbc;
RSQL_ERRCODE stat;
    ...
    stat = rsqlRegisterProc(hdbc, &findbook_defn);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlExecProc

rsqlRegisterUDFs

Register C-based user-defined functions

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlRegisterUDFs(
    HCONN               hConn,
    uint16_t            noUdfs,
    const UDFLOADTABLE *udftab)
```

Arguments

hConn	(input)	Connection handle.
noUdfs	(input)	Number of user-defined functions specified in udftab.
udftab	(input)	Pointer to the user-defined function load table.

Description

Function rsqlRegisterUDFs needs to be called in order to make the function entry points associated with a set of user-defined functions available to the RDMe SQL engine on the connection specified by hConn. The UDFLOADTABLE is declared in header file rsqltypes.h (which is #include'd by rdmsql.h). This function must be called prior to calling rsqlPrepare which compiles a statement that calls any of the user-defined functions specified in udftab.

Only one udftab can be registered in a connection. Any additional attempts to call rsqlRegisterUDFs after the first will return error code errREGISTERED.

Example

```
#include "rdmsql.h"

HCONN        hdbc;
RSQL_ERRCODE stat;

/* user-defined processing functions for aggregate function udfCount */
UDFCHECK    CntCheck;
UDFFUNC     CntFunc;
UDFINIT     CntInit;
UDFCLEANUP  CntCleanup;
UDFRESET    CntReset;

/* Table of user-defined functions for this module */
static UDFLOADTABLE UdfTable[ ] = {
```

```
/*   udfName    udfType   udfCall  udfCheck  udfInit  udfCleanup  udfReset */
   {"udfCount", tINTEGER, CntFunc, CntCheck, CntInit, CntCleanup, CntReset}
};

/* table size */
#define NFCNS (sizeof(UdfTable)/sizeof(UDFLOADTABLE))
...
       stat = rsqlRegisterUDFs(hdbc, NFCNS, &UdfTable);
...
       stat = rsqlPrepare(hstmt, "select mgrid, udfCount(patid) from patron group by
1");
...
```

RSQL User Guide

UDF Load Table Definition and Registration

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
78	errREGISTERED	42000	UDF/UDP/XTF already registered

See Also

rsqlRegisterVirtualTables

rsqlRegisterVirtualTables

Register C-based virtual tables

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlRegisterVirtualTables(
    HCONN                hConn,
    const char          *dbname,
    int16_t              noVtabs,
    const VTFLOADTABLE  *vtabs)
```

Arguments

`hConn`	(input)	Connection handle.
`dbname`	(input)	A pointer to a char string containing the name of the database in which the virtual tables are declared.
`noVtabs`	(input)	The number of virtual tables (the size of the `vtabs` array).
`vtabs`	(input)	A pointer to the virtual table function load table.

Description

Function `rsqlRegisterVirtualTables` must be called to make the function entry points associated with the virtual tables declared in database `dbname` available to the RDMe SQL engine on the connection identified by `hConn`. This function *must* be called after `hConn` has been allocated but *before* database `dbname` has been opened.

Only one call to `rsqlRegisterVirtualTables` can be made for any particular database on this connection. Any successive calls will return error code `errREGISTERED`.

Example

RDMe SQL uses virtual tables to implement the ODBC catalog access functions (e.g., **SQLTable**). The example below shows the code used to register the `syscat` virtual tables.

```
#include "rdmsql.h"

/* Function prototypes */
static VTINIT     vcatInit;
static VTROWCOUNT vtabRowCount;
static VTEXECUTE  vtabExecute;
static VTFETCH    vtabFetch;
static VTCLOSE    vtabClose;
static VTTERM     vcatTerm;
```

```
static VTROWCOUNT vcolRowCount;
static VTEXECUTE  vcolExecute;
static VTFETCH    vcolFetch;
static VTCLOSE    vcolClose;

static VTINIT     vplanInit;
static VTROWCOUNT vplanRowCount;
static VTEXECUTE  vplanExecute;
static VTFETCH    vplanFetch;
static VTCLOSE    vplanClose;

VTFLOADTABLE vcatFcnTable[] = {
    {"sys$table",  vcat-
Init,vtabRowCount,vtabExecute,vtabFetch,vtabClose,vcatTerm},
    {"sys$column",
vcatInit,vcolRowCount,vcolExecute,vcolFetch,vcolClose,vcatTerm},
    {"sys$plan",   vpla-
nInit,vplanRowCount,vplanExecute,vplanFetch,vplanClose,vcatTerm}
};
int16_t vcatNoFcns = sizeof(vcatFcnTable)/sizeof(VTFLOADTABLE);

       ...
       rsqlRegisterVirtualTables(hdbc, "syscat", vcatNoFcns, vcatFcnTable);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
78	errREGISTERED	42000	UDF/UDP/XTF already registered

rsqlSetAutoCommit

Set the auto commit status for the specified connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetAutoCommit(
    const HCONN    hConn,
    int16_t        autocommit)
```

Arguments

hConn	(input)	Connection handle.
autocommit	(input)	Auto commit mode: 1 to turn on, 0 to turn off.

Description

Function `rsqlSetAutoCommit` is used to set the auto commit mode for the connection associated with `hConn`. The autocommit argument should be set to 1 to turn auto commit on and 0 to turn auto commit off. When auto commit is enabled, each **insert**, **update**, and **delete** statement will begin a transaction that will be automatically committed upon successful execution or rolled back in the event of an error. Auto commit mode is disabled by default.

> Note that it is initially enabled in the RDMe SQL ODBC/JDBC API.

It is generally not recommended that you operate with auto commit mode turned on because most database modifications involve related changes to multiple tables such that logical database consistency can only be guaranteed when those changes are made within a single transaction.

Example

```
#include "rdmsql.h"
...
HCONN   hdbc;
int16_t auto_commit_flag;
    ...
    /* turn off auto commit mode if enabled */
    rsqlGetAutoCommit(&auto_commit_flag, hdbc);
    if ( auto_commit_flag )
        rsqlSetAutoCommit(hdbc, 0);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value
36	errTRACTIVE	25000	transaction is active
162	errLOCKMODE	RX026	illegal locking mode

See Also

rsqlGetAutoCommit

rsqlSetCursorName

Set the cursor name for the specified statement handle

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetCursorName(
    HSTMT       hStmt,
    char       *pCursor
)
```

Arguments

`hStmt`	`(input)`	Statement handle.
`pCursor`	`(input)`	A pointer to a string containing the cursor name.

Description

Call function `rsqlSetCursorName` to set the current cursor name for the updateable **select** statement that has been compiled under statement handle `hStmt`. Cursor names are used with a positioned **delete** or **update** statements to make modifications to the current row of the associated **select** statement. The maximum string size allowed for a cursor name is `NAMELEN-1` (31 characters).

Example

```
#include "rdmsql.h"
...
HSTMT hstmt1, *hstmt2;
    ...
    rsqlPrepare(hstmt1, "select * from patron");
    rsqlSetCursorName(hstmt1, "my_cursor");
    rsqlPrepare(hstmt2, "delete from patron where current of my_cursor");
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
1	errTRUNCATE	01004	data truncation
12	errDUPCURSOR	3C000	duplicate cursor name
29	errINVNULL	HY009	invalid use of null pointer
52	errCURSTATE	24000	invalid cursor state
53	errNOTSELECT	24000	current statement is not SELECT

See Also

rsqlGetCursorName

rsqlSetDateFormat

Set the date constant format for the connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetDateFormat(
    HCONN              hConn,
    int16_t            datefmt)
```

Arguments

hConn	(input)	The connection handle.
datefmt	(input)	Format specifier: 1=MMDDYYYY, 2=YYYYMMDD, 3=DDMMYYYY.

Description

Call `rsqlSetDataFormat` to specify the order in which the month, day, and year values are to be interpreted by SQL in literal date constants for the connection specified by `hConn`. The format is indicated by the code specified in argument format. The following constants have been defined in the standard RDMe SQL header files for use with this function. The default date format is YYYYMMDD.

```
#define MMDDYYYY 1
#define YYYYMMDD 2     /* Default format */
#define DDMMYYYY 3
```

Of course the "DD" represents the day, "MM" the month, and "YYYY" the year. Thus, July 22, 2011 would be represented as follows for each of the three possible formats:

Format 1: date "07-22-2011"
Format 2: date "2011-07-22"
Format 3: date "22-07-2011"

Example

```
#include "rdmsql.h"
...
HCONN hdbc;

       ...
       /* set date constant format to European standard */
       rsqlSetDateFormat(hdbc, DDMMYYYY);
       ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
135	errINVDATEFMT	42000	invalid date format

See Also

rsqlGetDateSeparator

rsqlGetDateFormat

rsqlSetDateSeparator

rsqlSetDateSeparator

Set the current date constant separator character for the connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetDateSeparator(
    HCONN               hConn,
    char                datesep)
```

Arguments

hConn	(input)	The connection handle.
datesep	(input)	Character symbol to be used as the date constant separator. Must be either a dash ('-') or a forward slash ('/') character.

Description

Call `rsqlSetDataSeparator` to specify the character symbol ('-' or '/') used to separate the month, day, and year values in data constants for the connection specified by `hConn`. The default date separator is '-'.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
       ...
       /* set date format to USA "standard" */
       rsqlSetDateFormat(hdbc, MMDDYYYY);

       /* set date format character to a slash */
       rsqlSetDateSeparator(hdbc, '/');

       printf("date constant format: date \"mm/dd/yyyy\"\n");
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
136	errINVDATESEP	42000	invalid date separator

See Also

rsqlGetDateFormat

rsqlSetDateFormat

rsqlGetDateSeparator

rsqlSetDeferBlobMode

Set a statement's deferred reading mode for blob data

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetDeferBlobMode(
    HSTMT    hStmt,
    int16_t  mode)
```

Arguments

hStmt	(input)	The statement handle.
mode	(input)	Specify 1 to enable deferred reading of blobs, specify 0 to disable deferred reading of blobs.

Description

This function can be called to enable or disable deferred reading of blob data. A blob column is a column of type **long varchar**, **long wvarchar** or **long varbinary**. When deferred reading is enabled the only way to retrieve the result value of a blob column returned in the result set from a `rsqlFetch` call on a **select** statement is through calls to `rsqlGetData`. This allows the blob data to be retrieved a block at a time. When deferred reading is disabled, the entire blob data value is read into an allocated buffer of type `tCHAR` (for **long varchar** columns), `tWCHAR` (for **long wvarchar** columns), or `tBINARY` (for **long varbinary** columns) that is returned in the `RSQL_VALUE` array returned from the call to `rsqlFetch`.

By default, deferred blob reading mode is disabled. Deferred reading of blobs (and every result column, for that matter) is automatically assumed when `rsqlFetch` is called with a `NULL` result argument.

Example

```
#include "rdmsql.h"
...
RSQL_VALUE *row;
uint32_t    len;
char        blobtxt[66];
HSTMT       hstmt;
       ...
       /* enable deferred reading of blobs */
       rsqlSetDeferBlobMode(hconn, 1);
       rsqlExecDirect(hstmt, "select last_name, short_bio from author");
       while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
              printf("author: %s\n", row[0].vt.cv);
              /* retrieve blob value in blocks of up to 65 bytes at a time */
```

```
                while ( rsqlGetData(hstmt, 2, &row[1], 65, &len) == errSUCCESS ) {
                        /* call app function to format blob text line */
                        FormatBlob(&row[1], len, blobtxt);
                        printf("      : %s\n", blobtxt);
                }
        }
        ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
28	errINVARG	HY009	invalid argument value

See Also

rsqlFetch

rsqlGetData

rsqlGetDeferBlobMode

rsqlSetErrorCallback

Set an RDMe SQL error callback function

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetErrorCallback(
       HCONN               hConn,
       RSQL_ERRHANDLER     errfcn,
       void               *data)
```

Arguments

`hConn`	`(input)`	The connection handle.
`errfcn`	`(input)`	User error function to be called by RDMe SQL when an error occurs.
`data`	`(input)`	Pointer to a user data area that will be passed by SQL to the error handler function when an error occurs.

Description

This function registers an application's error handling function that will be called by RDMe SQL whenever an error occurs. SQL will call the registered function after it has completed all of its own error handling and clean up.

All errors except `errINVHANDLE` will cause the error handling function to be called. Additionally, errors returned from the following functions cannot call the error handling function:

- rsqlTFSInit
- rsqlAllocConn
- rsqlGetErrorMsg
- rsqlGetErrorInfo
- rsqlSetErrorCallback

The error handling function that is passed into `rsqlSetErrorCallback` must be declared to exactly match the following prototype:

Error Handler Prototype

```
RSQL_ERRCODE EXTERNAL_FCN MyErrorHandler(
    HRSQL           hndl,
    RSQL_ERRCODE    stat,
    void           *data)
```

Error Handler Arguments

`hndl`	`(input)`	The connection handle or the statement handle for which the error occurred.
`stat`	`(input)`	Error code returned from the most recent RDMe SQL API call.
`data`	`(input)`	Pointer to a user data area that will be passed by SQL to the error handler function when an error occurs.

The contents of the data area is totally up to the developer. Typical uses would be to include a statement handle, connection handle, error count (see example below).

The error handler can be disabled by calling `rsqlSetErrorCallback` with the `errfcn` argument set to NULL.

The error handler is free to use a `longjmp` call to exit out of the context in which the RDMe SQL API function was called. You are allowed to call `rsql` functions from the error handler. Care should be taken to avoid infinite recursion as any error from these calls may invoke the error handler again.

The error code returned by the error handler will be delivered back to the API function for which the error happened. This error code will not be stored in the connection or statement handle. Querying any of these handles will yield the last error code set by the SQL engine.

The developer can use an error handler to get a full stack trace for your application and the SQL engine when errors happen. Set a break point from a debugger in the error handler. When the break point is hit the debugger can give you a stack trace. When appropriate, please attach such a stack trace if you file a bug.

Example

```
#include "rdmsql.h"
...
struct mydata_t {
    char *myContext
};

#define MSGLEN 133

static RSQL_ERRCODE EXTERNAL_FCN MySQLErr (HRSQL hndl, RSQL_ERRCODE code, void
*data) {
    struct mydata_t *md= (struct mydata_t *)data;
    RSQL_ERRCODE status;
    char    errmsg[MSGLEN];

    status= rsqlGetErrorInfo(hndl, errmsg, MSGLEN);
    if (status == errSUCCESS) {
        printf("*** error: %s (%s)\n", errmsg, myContex);
        ...
    }
    return code;
}

... MyFunction (...) {
```

```
    struct mydata_t mydata;
    mydata.myContext = "MyApplication";

    rsqlSetErrorCallback (hdbc, MySQLErr, &mydata);
    ...
}
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error code returned from the most recent RDMe SQL API call

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlAllocConn

rsqlGetConnHandle

rsqlSetGenCFiles

Set the connection handle's "generate C files" mode

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetGenCFiles(
    const HCONN    hConn,
    int16_t        GenCFiles)
```

Arguments

hConn	(input)	Connection handle.
GenCFiles	(input)	Generate C files mode: 1=enable, 0=disable.

Description

This function controls the feature of RDMe SQL to generate the C Language source files when compiling a SQL database schema. When the feature is enabled, the SQL schema compilation will generate a set of C Language source files associated with the database defined in the schema. The generated source files will be placed in the current working directory of the process that compiled the SQL schema.

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
int16_t gencfile_flag;
    ...
    /* turn off generate C file mode if enabled */
    rsqlGetGenCFiles(&gencfile_flag, hdbc);
    if ( gencfile_flag )
        rsqlSetGenCFiles(hdbc, 0);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlGetGenCFiles

rsqlSetReadOnlyTrmode

Set the current read only transaction mode setting

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetReadOnlyTrmode(
    HCONN               hConn,
    int16_t             romode)
```

Arguments

hConn	(input)	The connection handle.
romode	(input)	New mode setting: 0 = manual, 1 = auto.

Description

You can call `rsqlSetReadOnlyTrmode` to assign the current read only transaction mode. When `romode` is 1, SQL will process all **select** statements within a system-generated read only transaction. When `romode` is set to 0, SQL will first issue read lock requests for the tables referenced in the **select** statement.

Example

```
#include "rdmsql.h"
    ...
void *hdbc;
    ...
    rsqlSetReadOnlyTrmode(hdbc, 1);
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
28	errINVARG	HY009	invalid argument value
-5	errINVHANDLE	02002	invalid connection or statement handle

See Also

rsqlGetReadOnlyTrmode

["Read Only Transactions" in "Concurrent Database Access" chapter]

rsqlSetTimeout

Set a connection's lock request timeout value

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlSetTimeout(
    HCONN           hConn,
    int32_t         secs
)
```

Arguments

`hConn`	`(input)`	The connection handle.
`secs`	`(input)`	The number of seconds to wait before timing out.

Description

Function `rsqlSetTimeout` can be called to specify the number of seconds that any ungranted lock request made from the connection associated with `hConn` can wait on the lock request queue. The number of seconds to wait is specified by argument `secs`. A negative value indicates that no lock requests will timeout (i.e., the request will wait indefinitely). This is not recommended unless you are very certain that your use of locking is deadlock free. A value of 0 will cause lock requests to immediately timeout if the lock cannot be granted. The system default timeout is 10 seconds.

When a timeout occurs, the SQL statement execution that initiated the lock request will return status `errTIMEOUT`. The proper procedure for dealing with timeouts is to free all held locks and restart the operation. If the timeout occurs within a transaction then the transaction should be rolled-back and the transaction restarted. If the timeout occurs outside a transaction then all held locks should be freed (**rollback** or **commit** outside a transaction will free all read locks).

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
RSQL_ERRCODE stat;
int32_t  to_secs = 25;
    ...
    /* reset lock request timeout to 25 seconds */
    rsqlSetTimeout(hdbc, to_secs);

    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected

SQL User's Guide

Locking in RDMe SQL

rsqlTFSInit

Initialize RDMe SQL Transactional File Server interaction.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTFSInit(
       const char       *docroot,
       const TFS_PARAMS *tparams)
```

Arguments

`docroot`	`(input)`	The path specification for the directory under which all databases controlled by the TFS are stored. If `docroot` is NULL, "", or "." then the current directory will be used as the document root.
`tparams`	`(input)`	Pointer to the TFS operational parameters descriptor packet. If `tparams` is NULL then system defaults will be used.

Description

Call this function before any calls to `rsqlAllocConn` to set up a document root other than the default, with other parameters than the default, or where all connections need to use the same TFS instance. You will need to call `rsqlTFSTerm` after the last connection have disconnected.

> This function does not have any effect if the remote TFS is used (TFSR).

The `docroot` string provides the path specification for the directory under which all databases that are to be controlled by this TFS are stored. If `docroot` is `NULL`, "", or "." then the current directory will be assumed to be the document root directory.

The `TFS_PARAMS` struct has several fields but the fields in the following table should be assigned the values specific to your particular application requirements.

TFS_PARAMS Structure Definition

Field	Type	Description
`no_disk`	`uint32_t`	Set to 1 if diskless, otherwise (normal) set to 0 (default)
`rd_only`	`uint32_t`	Set to 1 if on a read only device, otherwise (normal) set to 0 (default)
`dbkeep`	`uint16_t`	[Windows only] Set to 1 (default) to run the dbkeep module that prevents in-memory segments from being destroyed.

Do not assign these fields using a struct initialization but explicitly use an assignment statement to assign the values for each field as in the example below. It is also a good idea to first zero out the `TFS_PARAMS` struct variable to clear the unused or internal use fields. Note that if you are not running diskless nor on a read-only device, simply pass `NULL` for the `tparams` argument.

Example

```
#include "rdmsql.h"
    ...
RSQL_ERRCODE stat;
const char *docroot = "c:\rdme_dbs";
const TFS_PARAMS tfspars;
HCONN hdbc;
    ...
    memset(&tfspars, 0, sizeof(TFS_PARAMS));
    tfspars.no_disk = 1;  /* running diskless */
    tfspars.rd_only = 0;  /* updates are allowed */
    tfspars.dbkeep = 1;   /* okay, even if not running Windows */

    stat = rsqlTFSInit(docroot, &tfspars);
    if ( stat == errSUCCESS ) {
        rsqlAllocConn(&hdbc);
        ...
        rsqlTFSTerm();
    }
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
154	errTFSFAILURE	RX024	unable to connect to TFS

See Also

rsqlTFSTerm

rsqlTFSTerm

Terminate RDMe SQL Transactional File Server interaction.

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTFSTerm( void )
```

Description

This function frees resources allocated by the TFS during and after the call to `rsqlTFSInit`.

> This function does not have any effect if the remote TFS is used (TFSR).

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
154	errTFSFAILURE	RX024	unable to connect to TFS

See Also

rsqlTFSInit

rsqlTransCommit

Commit transaction's changes to database

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransCommit(
    HCONN           hConn)
```

Arguments

`hConn`	`(input)`	The connection handle.

Description

A call to function `rsqlTransCommit` will commit the changes within the connection specified by `hConn` made since the start of the transaction to the database. Upon successful return from `rsqlTransCommit` the transaction's changes are guaranteed to have been safely written to the database files. All locks that were issued during the transaction are freed.

If `rsqlTransCommit` is called outside of a transaction all of the locks currently held on connection `hConn` will be freed.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStart(hdbc, "AddAccount");

    ...                 /* insert new account record into database */

    if ( AccountError() )
        rsqlTransRollback(hdbc, "AddAccount");
    else {
        rsqlTransSavepoint(hdbc, "CreditCard");

        ...             /* get and store credit card info in database */

        if ( ValidCreditCard() )
            rsqlTransRelease(hdbc, "CreditCard");
        else {
            ...                 /* report error and mark status in account record */
```

```
            /* discard credit card data but keep account intact */
            rsqlTransRollback(hdbc,"CreditCard");
        }
    }
    rsqlTransCommit(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
41	errDBNOTOPEN	42000	database not open

See Also

rsqlTransStart

rsqlTransStartReadOnly

rsqlTransEndReadOnly

rsqlTransRollback

rsqlTransEndReadOnly

End a read-only transaction

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransEndReadOnly(
    HCONN           hConn)
```

Arguments

hConn	(input)	The connection handle.

Description

Call function `rsqlTransEndReadOnly` to end a read-only transaction on the connection associated with `hConn`. Read-only transactions take a transaction-consistent "snapshot" of the database at the time the read-only transaction begins. Locking is unnecessary within a read-only transaction so other processes that are making changes to the database will not be blocked. However, those modifications made by other users will not be visible in this connection until the read-only transaction completes.

It is important that you try and keep read-only transactions short because a long running read-only transaction can result in a degradation of system performance.

Error code errRDONLYFLAG is returned when `rsqlTransEndReadOnly` is called when a read-only transaction is not active.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStartReadOnly(hdbc);

    ...                 /* issue any number of SELECT statements */

    rsqlTransEndReadOnly(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
37	errTRNOTACT	25000	transaction not active
167	errRDONLYFLAG	25000	Inconsistent read-only transaction commit/rollback/end call

See Also

rsqlTransStartReadOnly

rsqlTransRelease

Release (delete) a transaction savepoint

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransRelease(
    HCONN           hConn,
    char           *txid)
```

Arguments

hConn	(input)	The connection handle.
txid	(input)	transaction id string (can be NULL).

Description

Call function rsqlTransRelease to discard a savepoint that was created by a prior call to rsqlTrans-Savepoint with a matching transaction id string, txid, within the active transaction associated with connection handle hConn.

Savepoints are automatically released on a call to rsqlTransCommit or any call to rsqlTransRollback that would either rollback changes to a prior savepoint or all of the changes made by the transaction. However, as soon as it is known that a rollback to a given savepoint is no longer necessary and if more changes within the transaction are possible, the savepoint should be released by calling rsqlTransRelease.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStart(hdbc, "AddAccount");

    ...                 /* insert new account record into database */

    rsqlTransSavepoint(hdbc, "CreditCard");

    ...                 /* get and store credit card info in database */

    if ( ValidCreditCard() )
       rsqlTransRelease(hdbc, "CreditCard");
    else {
       ...              /* report error and mark status in account record */
```

```
        /* discard credit card data but keep account intact */
        rsqlTransRollback(hdbc,"CreditCard");
    }
    rsqlTransCommit(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
37	errTRNOTACT	25000	transaction not active
129	errROTRACT	25000	read-only transaction is active

See Also

rsqlTransSavepoint

rsqlTransRollback

Rollback (discard) database changes made during a transaction

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransRollback(
    HCONN           hConn,
    char           *txid)
```

Arguments

`hConn`	`(input)`	The connection handle.
`txid`	`(input)`	transaction id string (can be `NULL`).

Description

Call function `rsqlTransRollback` to rollback (i.e., undo, discard) the changes made after either the start of the transaction or since the savepoint with the matching transaction id, `txid`, that was created by a prior call to `rsqlTransSavepoint`.

A rollback of an entire transaction automatically frees all of the locks that were issued during the transaction.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStart(hdbc, "AddAccount");

    ...                 /* insert new account record into database */

    if ( AccountError() )
       rsqlTransRollback(hdbc, "AddAccount");
    else {
       rsqlTransSavepoint(hdbc, "CreditCard");

       ...              /* get and store credit card info in database */

       if ( ValidCreditCard() )
              rsqlTransRelease(hdbc, "CreditCard");
       else {
              ...              /* report error and mark status in account record */
```

```
                /* discard credit card data but keep account intact */
                rsqlTransRollback(hdbc,"CreditCard");
            }
        }
        rsqlTransCommit(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
174	errINVTRID	RX027	invalid transaction id

See Also

rsqlTransStart

rsqlTransRelease

rsqlTransSavepoint

rsqlTransSavepoint

Start a transaction

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransSavepoint(
    HCONN           hConn,
    char           *txid)
```

Arguments

hConn	(input)	The connection handle.
txid	(input)	transaction savepoint id string (can be NULL).

Description

Function rsqlTransSavepoint can be called to mark a savepoint within a database modification transaction on the connection associated with hdbc. The database changes that are made after the savepoint can be rolled back by a subsequent call to rsqlTransRollback that has a transaction id, txid, that matches the one specified in this call to rsqlTransSavepoint. The changes that were made prior to this call to the rsqlTransSavepoint whose changes are to be rolled back are unaffected.

Any number of savepoints can be initiated within a given transaction. However, a rsqlTransRollback to any particular savepoint will discard all of the savepoints that were issued after it.

Savepoints remain in place until either a transaction commit, rollback or rollback to the savepoint, or until the savepoint is released by a call to rsqlTransRelease. As soon as it is known that a rollback to the savepoint is no longer necessary and if more changes within the transaction are possible, the savepoint should be released by calling rsqlTransRelease.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStart(hdbc, "AddAccount");

    ...                 /* insert new account record into database */

    if ( AccountError() )
       rsqlTransRollback(hdbc, "AddAccount");
    else {
```

```
        rsqlTransSavepoint(hdbc, "CreditCard");

        ...             /* get and store credit card info in database */

        if ( ValidCreditCard() )
                rsqlTransRelease(hdbc, "CreditCard");
        else {
                ...             /* report error and mark status in account record */

                /* discard credit card data but keep account intact */
                rsqlTransRollback(hdbc,"CreditCard");
        }
    }
    rsqlTransCommit(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer
37	errTRNOTACT	25000	transaction not active
129	errROTRACT	25000	read-only transaction is active

See Also

rsqlTransRollback

rsqlTransRelease

rsqlTransStart

Start a transaction

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransStart(
    HCONN           hConn,
    char           *txid)
```

Arguments

`hConn`	`(input)`	The connection handle.
`txid`	`(input)`	transaction id string (can be `NULL`).

Description

Function `rsqlTransStart` can be called to start a new database modification transaction on the connection associated with `hConn`. The transaction id, `txid`, is a string that can be specified to identify or name a specific transaction. The string value is completely arbitrary and up to you to decide how, if at all, you want to use it.

If called when auto commit mode is enabled, auto commit mode will be temporarily disabled until the transaction completes (i.e., either commits or rolled back).

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStart(hdbc, "AddAccount");

    ...                 /* insert new account record*/

    rsqlTransCommit(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
36	errTRACTIVE	25000	transaction is active
129	errROTRACT	25000	read-only transaction is active

See Also

rsqlTransCommit

rsqlTransRollback

rsqlTransStartReadOnly

rsqlTransEndReadOnly

rsqlTransStartReadOnly

Start a read-only transaction

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransStartReadOnly(
    HCONN           hConn)
```

Arguments

`hConn`	`(input)`	The connection handle.

Description

Function `rsqlTransStartReadOnly` can be called to start a read-only transaction on the connection associated with `hConn`. Read-only transactions take a transaction-consistent "snapshot" of the database at the time the read-only transaction begins. Locking is unnecessary within a read-only transaction so other processes that are making changes to the database will not be blocked. However, those modifications made by other users will not be visible in this connection until the read-only transaction completes.

It is important that you try and keep read-only transactions short because a long running read-only transaction can result in a degradation of system performance.

Example

```
#include "rdmsql.h"
    ...
HCONN hdbc;
    ...
    rsqlTransStartReadOnly(hdbc);

    ...                 /* issue any number of SELECT statements */

    rsqlTransEndReadOnly(hdbc);
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
41	errDBNOTOPEN	42000	database not open
36	errTRACTIVE	25000	transaction is active
129	errROTRACT	25000	read-only transaction is active
132	errLOCKSACTIVE	25000	operation not allowed due to active read locks

See Also

rsqlTransEndReadOnly

rsqlTransStatus

Return the current transaction state for the specified connection

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlTransStatus(
    HCONN            hConn,
    TRANS_STAT      *pTrstat)
```

Arguments

`hConn`	`(input)`	The connection handle.
`pTrstat`	`(output)`	Pointer to variable into which the transaction status will be returned.

Description

This function can be called to determine the transaction status condition on the specified connection. The status is returned in `*pTrstat` and will have one of the values from the following table:

Transaction Status Constants

TRANS_STAT	Value	Description
`transINACTIVE`	0	No transaction is active
`transACTIVE`	1	A standard (modification) transaction is active
`transREADONLY`	2	A read-only transaction is active

Example

```
#include "rdmsql.h"
...
HCONN hdbc;
TRANS_STAT trstat;
       ...
       rsqlTransStatus(hdbc, &trstat);
       switch ( trstat ) {
               case transINACTIVE: printf("no transaction is active\n"); break;
               case transACTIVE  : printf("a modification transaction is active\n"); break;
               case transREADONLY: printf("a read-only transaction is active\n"); break;
       }
..
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlTransStart

rsqlTransStartReadOnly

rsqlUnlockTable

Free a read lock on a database table

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUnlockTable(
    HCONN           hConn,
    const char     *dbname,
    const char     *tabname)
```

Arguments

hConn	(input)	The connection handle.
dbname	(input)	Pointer to string containing the name of the database containing the table to be unlocked.
tabname	(input)	Pointer to string containing the name of the table to be unlocked.

Description

Call `rsqlUnlockTable` to free a read lock on table `tabname` in database `dbname`. This function will only unlock a read locked table outside of a transaction. Locks within a transaction are necessarily held until the transaction is either committed or rolled back.

Example

```
#include "rdmsql.h"
...
RSQL_ERRCODE stat;
HCONN *hdbc;
TABLE_LOCK lock_request[] = {{"author", lockREAD}};
    ...
    stat = rsqlLockTables(hdbc, "bookshop", 1, lock_request);
       if ( stat == errSUCCESS ) {
       ... /* issue and process SELECT statement on author table */
       rsqlUnlockTable(hdbc, "bookshop", "author");
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
14	errTABNOTFOUND	42S02	table not found
29	errINVNULL	HY009	invalid use of null pointer
41	errDBNOTOPEN	42000	database not open
129	errROTRACT	25000	read-only transaction is active
130	errUNLOCKINTRANS	25000	unlock not allowed in a transaction
131	errNOTLOCKED	25000	table is not locked
162	errLOCKMODE	RX026	illegal locking mode

SQL User's Guide

Locking in RDMe SQL

See Also

rsqlLockTables

rsqlUnlockTableAll

rsqlUnlockTableAll

Unlock all read locked tables

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUnlockTableAll(
    HCONN           hConn)
```

Arguments

hConn	(input)	The connection handle.

Description

A call to function `rsqlUnlockTableAll` will free read locks on all of the tables that have been locked on the specified connection through prior calls to `rsqlLockTable`. This function will only unlock read locked tables outside of a transaction. Locks within a transaction are necessarily held until the transaction is either committed or rolled back.

Example

```
#include "rdmsql.h"
...
RSQL_ERRCODE stat;
HCONN *hdbc;
TABLE_LOCK lock_request[] = {
       {"author", lockREAD},
       {"book", lockREAD}
};
int16_t notabs = sizeof(lock_request)/sizeof(TABLE_LOCK);
    ...
    stat = rsqlLockTables(hdbc, "bookshop", notabs, lock_request);
    if ( stat == errSUCCESS ) {
       ... /* issue and process SELECT statement on author and book tables */
    rsqlUnlockTableAll(hdbc);
    }
    ...
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
`rdmersql10`	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
130	errUNLOCKINTRANS	25000	unlock not allowed in a transaction
162	errLOCKMODE	RX026	illegal locking mode

SQL User's Guide

Locking in RDMe SQL

See Also

rsqlLockTables

rsqlUnlockTable

rsqlUnpackDate

Unpack a binary DATE_VAL into a CAL_DATE structure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUnpackDate(
    DATE_VAL       *pPDt,
    CAL_DATE       *pUDt)
```

Arguments

pPDt	(input)	Pointer to variable containing packed date value.
pUDt	(output)	Pointer variable to contained unpacked date value.

Description

Call `rsqlUnpackDate` to convert the packed `DATE_VAL` value pointed to by `pPDt` into an unpacked date value which will be returned into the `CAL_DATE struct` variable pointed to by `pUDt`. The `CAL_DATE struct` and `DATE_VAL typedef` are given below.

```
typedef uint32_t DATE_VAL;

typedef struct tagCAL_DATE {
    int32_t  year;      /* 1 - ???? */
    uint16_t month;     /* 1 - 12 */
    uint16_t day;       /* 1 - 31 */
} CAL_DATE;
```

This function is needed to unpack date values from the packed `tDATE` format when retrieving `RSQL_VALUE` date result column values via calls to `rsqlFetch` and `rsqlGetData`.

The error messages associated with the error codes returned by this function can only be retrieved by calling `rsqlGetErrorMsg` as `rsqlGetErrorInfo` only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_ERRCODE stat;
RSQL_VALUE *row;

CAL_DATE  acqndt, solddt;
        ...
        rsqlExecDirect(hstmt, "select date_acqd, date_sold, bookid from book order by
1");
        while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
                rsqlUnpackDate(row[0].vt.dtv, &acqndt);
                rsqlUnpackDate(row[1].vt.dtv, &solddt);
                printf("acquired: %02d/%02d/%d sold: %02d/%02d/%d id: %s\n",
                        acqndt.month, acqndt.day, acqndt.year,
                        solddt.month, solddt.day, solddt.year,
                        row[2].vt.cv)
        }
        ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
15	errDATETIMEOVF	22008	date/time value overflow
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlPackDate

rsqlUnpackTime

rsqlUnpackTimestamp

rsqlUnpackTime

Unpack a binary TIME_VAL into a CAL_TIME structure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUnpackTime(
    TIME_VAL       *pPTm,
    CAL_TIME       *pUTm)
```

Arguments

pPTm	(input)	Pointer to variable containing packed time value.
pUTm	(output)	Pointer variable to contained unpacked time value.

Description

Call rsqlUnpackTime to convert the packed TIME_VAL value pointed to by pPTm into an unpacked time value which will be returned into the CAL_TIME struct variable pointed to by pUTm. The CAL_TIME struct and TIME_VAL typedef are given below.

```
typedef uint32_t TIME_VAL;

typedef struct tagCAL_TIME {
    uint16_t hour;      /* 0 - 23 */
    uint16_t minute;    /* 0 - 59 */
    uint16_t second;    /* 0 - 59 */
    uint16_t fraction;  /* 0 - 9999 */
} CAL_TIME;
```

This function is needed to unpack time values from the packed tTIME format when retrieving RSQL_VALUE time result column values via calls to rsqlFetch and rsqlGetData.

The error messages associated with the error codes returned by this function can only be retrieved by calling rsqlGetErrorMsg as rsqlGetErrorInfo only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_VALUE *row;
CAL_TIME  last_bid_time;
        ...
        rsqlExecDirect(hstmt, "select bid_time, aucid, bookid from auction order by 1
desc");
        while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
                rsqlUnpackTime(row[0].vt.tmv, &last_bid_time);
                printf("last bid time: %02d:%02d:%02d auction id: %s bookid: %s\n",
                        last_bid_time.hour, last_bid_time.minute, last_bid_time.second,
                        row[1].vt.cv, row[2].vt.cv);
        }
        ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
15	errDATETIMEOVF	22008	date/time value overflow
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlPackTime

rsqlUnpackDate

rsqlUnpackTimestamp

rsqlUnpackTimestamp

Unpack a binary TIMESTAMP_VAL into a CAL_TIMESTAMP structure

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUnpackTimestamp(
    TIMESTAMP_VAL  *pPTs,
    CAL_TIMESTAMP  *pUTs)
```

Arguments

pPTs	(input)	Pointer to variable containing packed timestamp value.
pUTs	(output)	Pointer variable to contained unpacked timestamp value.

Description

Call `rsqlUnpackTimestamp` to convert the packed `TIMESTAMP_VAL` value pointed to by `pPTs` into an unpacked time value which will be returned into the `CAL_TIMESTAMP struct` variable pointed to by `pUTs`. The `CAL_TIMESTAMP struct` and `TIMESTAMP_VAL typedefs` are given below.

```
typedef struct {
    DATE_VAL date;
    TIME_VAL time;
} TIMESTAMP_VAL;

typedef struct tagCAL_TIMESTAMP {
    int32_t  year;      /* 1 - ???? */
    uint16_t month;     /* 1 - 12 */
    uint16_t day;       /* 1 - 31 */
    uint16_t hour;      /* 0 - 23 */
    uint16_t minute;    /* 0 - 59 */
    uint16_t second;    /* 0 - 59 */
    uint16_t fraction;  /* 0 - 9999 */
} CAL_TIMESTAMP;
```

This function is needed to unpack timestamp values from the packed `tTIMESTAMP` format when retrieving `RSQL_VALUE` timestamp result column values via calls to `rsqlFetch` and `rsqlGetData`.

The error messages associated with the error codes returned by this function can only be retrieved by calling `rsqlGetErrorMsg` as `rsqlGetErrorInfo` only returns error info associated with connection or statement handles.

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Example

```
#include "rdmsql.h"
...
HSTMT hstmt;
RSQL_VALUE *row;
CAL_TIMESTAMP  when_bid;
        ...
        rsqlExecDirect(hstmt, "select bid_ts, aucid, patid from bid order by 1 desc");
        while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
                rsqlUnpackTimestamp(row[0].vt.tsv, &when_bid);
                printf("bid time: %02/%02/%d @%02d:%02d:%02d auction id: %s patid: %s\n",
                        when_bid.month, when_bid.day, when_bid.year,
                        when_bid.hour, when_bid.minute, when_bid.second,
                        row[1].vt.cv, row[2].vt.cv);
        }
        ...
```

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
29	errINVNULL	HY009	invalid use of null pointer

See Also

rsqlPackTimestamp

rsqlUnpackDate

rsqlUnpackTime

rsqlUpdateRow

Store the updated column values for the current row

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUpdateRow(
    HSTMT               hStmt)
```

Arguments

hStmt	(input)	Statement handle associated with the **select** statement whose rows are currently being fetched (rsqlFetch).

Description

Function rsqlUpdateRow must be called to stored the updated column values made by prior calls to rsqlUpdateCol for the **select** statement result set whose rows are currently being fetched via calls to rsqlFetch.

Any errors caused by the update column values will be detected at this time.

The table being updated will request to have its read lock upgraded to a write lock on the first call to rsqlUpdateRow within the transaction. So, it is possible that the subsequent call to rsqlUpdateRow can return status errTIMEOUT. In order to avoid this situation, you may wish to explicitly write lock the table (call rsqlLockTable) prior to the execution of the **select** statement. If the upgrade is granted the table will remain locked while the cursor on that **select** statement remains open and the transaction remains active. When the transaction is either committed or rolled back the cursor is automatically closed so that subsequent fetches are no longer allowed.

Note that use of rsqlUpdateCol/rsqlUpdateRow cannot be used when autocommit transaction mode is enabled.

Example

```
#include "rdmsql.h"

main()
{
    HCONN hdbc;
    HSTMT hstmt;
    RSQL_ERRCODE stat;
    RSQL_VALUE *row, newcomm;
```

```
    rsqlAllocConn(&hdbc);
    rsqlOpenDB(hdbc, "bookshop", "s");
    rsqlAllocStmt(hdbc, &hstmt);

    newcomm.type = tDOUBLE;
    newcomm.len = 0;
    rsqlExecDirect(hstmt, "start transaction");
    rsqlExecDirect(hstmt, "select mgrid, commission from acctmgr for update");

    while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
        if ( row[1].vt.dv >= 0.03 ) {
            printf("changing commission for %s from %.2f ",
                                    row[0].vt.cv, row[1].vt.dv);
            if ( row[1].vt.dv >= 0.05 )
                newcomm.vt.dv = row[1].vt.dv + 0.05;
            else if ( row[1].vt.dv >= 0.03 )
                newcomm.vt.dv = row[1].vt.dv + 0.04;
            else
                newcomm.vt.dv = row[1].vt.dv + 0.01;
            printf("to %.2f\n", newcomm.vt.dv);
            rsqlUpdateCol(hstmt, 2, &newcomm);
            stat = rsqlUpdateRow(hstmt);
            if ( stat != errSUCCESS )
                printf("problem updating %s\n", row[0].vt.cv);
        }
    }
    rsqlExecDirect(hstmt, "commit");
}
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
163	errNOUPDCOLS	42000	no columns have been updated

See Also

rsqlUpdateCol

rsqlCancelRow

rsqlUpdateCol

Update the value for one of the **select** statement result set columns

Prototype

```
RSQL_ERRCODE EXTERNAL_FCN rsqlUpdateCol(
       HSTMT                    hStmt,
       uint16_t                 colno,
       const RSQL_VALUE        *value)
```

Arguments

hStmt	(input)	Statement handle associated with the **select** statement whose rows are currently being fetched (rsqlFetch).
colno	(input)	The number of the result set column whose value is to be updated.
value	(output)	A pointer to a RSQL_VALUE pointer variable containing the new column value.

Description

Function rsqlUpdateCol can be used to update a column from the **select** statement result set whose rows are currently being fetched via calls to rsqlFetch.

The select statement result column whose value is to be updated is specified by the column number, colno which must be a value between 1 and the number of result columns in the **select** statement. The **select** statement must have been specified with the for update clause and the column identified by colno must be updateable.

The changes made by one or more calls to rsqlUpdateCol do not go into effect until function rsqlUpdateRow is called. Any errors caused by the update column value will be detected at that time.

The table being updated will request to have its read lock upgraded to a write lock on the first call to rsqlUpdateRow within the transaction. So, it is possible that the subsequent call to rsqlUpdateRow can return status errTIMEOUT. In order to avoid this situation, you may wish to explicitly write lock the table (call rsqlLockTable) prior to the execution of the **select** statement. If the upgrade is granted the table will remain locked while the cursor on that **select** statement remains open and the transaction remains active. When the transaction is either committed or rolled back the cursor is automatically closed so that subsequent fetches are no longer allowed.

Note that use of rsqlUpdateCol/rsqlUpdateRow cannot be used when autocommit transaction mode is enabled.

Example

```
#include "rdmsql.h"

main()
{
    HCONN hdbc;
    HSTMT hstmt;
    RSQL_ERRCODE stat;
    RSQL_VALUE *row, newcomm;


    rsqlAllocConn(&hdbc);
    rsqlOpenDB(hdbc, "bookshop", "s");
    rsqlAllocStmt(hdbc, &hstmt);

    newcomm.type = tDOUBLE;
    newcomm.len = 0;
    rsqlExecDirect(hstmt, "start transaction");
    rsqlExecDirect(hstmt, "select mgrid, commission from acctmgr for update");

    while ( rsqlFetch(hstmt, &row, NULL) == errSUCCESS ) {
        if ( row[1].vt.dv >= 0.03 ) {
            printf("changing commission for %s from %.2f ",
                                    row[0].vt.cv, row[1].vt.dv);
            if ( row[1].vt.dv >= 0.05 )
                newcomm.vt.dv = row[1].vt.dv + 0.05;
            else if ( row[1].vt.dv >= 0.03 )
                newcomm.vt.dv = row[1].vt.dv + 0.04;
            else
                newcomm.vt.dv = row[1].vt.dv + 0.01;
            printf("to %.2f\n", newcomm.vt.dv);
            rsqlUpdateCol(hstmt, 2, &newcomm);
            stat = rsqlUpdateRow(hstmt);
            if ( stat != errSUCCESS )
                printf("problem updating %s\n", row[0].vt.cv);
        }
    }
    rsqlExecDirect(hstmt, "commit");
}
```

Required Headers

```
#include "rdmsql.h"
```

Libraries

Library Name	Description
rdmersql10	RSQL API Library

See Library Naming Conventions section for full library name and a list of library dependencies.

Return Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-5	errINVHANDLE	02002	invalid connection or statement handle
33	errCOLNUMBER	07009	invalid descriptor index (column number)
53	errNOTSELECT	24000	current statement is not SELECT
65	errINVSTATE	RX008	invalid statement state
139	errNOTUPDATEABLE	42000	SELECT or column is not updateable
164	errAUTOCOMMIT	42000	operation not allowed when autocommit is enabled

See Also

rsqlUpdateRow

rsqlCancelRow

SQL API Error/Status Codes

To err is human, but to really foul things up requires a computer.
- Farmers Almanac, 1978

Table 31. RDMe SQL API Status Codes

Error Code	Enum Identifier	SQL State	Description
0	errSUCCESS	00000	no error was detected
-1	errNOMOREDATA	02000	no more data
-2	errSQLERROR	RX002	internal SQL error
-3	errIGNORE	RX001	ignore this error
-4	errNEEDDATA	01000	need data
-5	errINVHANDLE	02002	invalid connection or statement handle

Note that numeric values for the error codes in the following tables is subject to change. In particular, the values for `errSYSTEM` will surely change as it is required to be at the end of the table. As new error codes are defined they are added to the table just prior to `errSYSTEM`

Table 32. RDMe SQL API Error Codes by Name

Error Code	Enum Identifier	SQL State	Description
151	errACCURANGE	HY101	Accuracy option type out of range
59	errATTRVALUE	HY024	invalid attribute value
164	errAUTOCOMMIT	42000	operation not allowed when autocommit is enabled
113	errAUTODESCR	HY017	invalid use of implicit descriptor handle
20	errBADBINLITERAL	22005	bad binary literal specification
144	errBADBLOBPAR	RX023	data-at-exec param type not compatible with blob (long var...) column
104	errBADCTYPE	07006	invalid C data type
31	errBADDATALEN	HY090	invalid string or buffer length
114	errBADDESCINFO	HY021	inconsistent descriptor information
9	errBADFORKEYCOL	42000	foreign key column is not declared in table
19	errBADFORMAT	42000	bad formatting specification
16	errBADKEYCOL	42000	key column not found
21	errBADLITERAL	22005	bad literal specification
76	errBADOUTERJOIN	42000	no access path between outer joined tables
6	errBADREFDCOL	42000	column is not declared in referenced table
105	errBADSTYPE	07006	invalid SQL data type
140	errBLOBDEFAULT	42000	default values not allowed on long var{char|binary} columns
141	errBLOBEXPR	42000	blobs cannot be referenced in expressions in deferred mode
143	errBLOBPARSONLY	RX022	data-at-exec params only allowed for blob (long

			var...) columns
102	errCANCELED	HY008	statement execution canceled by user
24	errCATNOTFOUND	42000	unable to open catalog file
171	errCHARREQD	42000	char/wchar type is required
153	errCIRCTABREF	42000	circular tables cannot be referenced
157	errCIRTABDELETE	42000	cannot delete rows from a circular table
57	errCLOSEHDBC	HY010	connection not closed
48	errCOLNOTFOUND	42S22	column name not found
33	errCOLNUMBER	07009	invalid descriptor index (column number)
147	errCOLRANGE	HY097	column type out of range
128	errCOMERROR	RX019	RPC communication error
145	errCONCATNULL	HY020	attempt to concatenate a null value
172	errCONNECT	08001	unable to connect
39	errCONNECTED	08000	must call before connect
51	errCURPOS	HY109	invalid cursor position
165	errCURSORTABLE	42000	positioned UPDATE/DELETE table does not match cursor's
52	errCURSTATE	24000	invalid cursor state
142	errDATAATEXEC	RX021	data-at-exec params only allowed with INSERT VALUES/UPDATE
55	errDATALOST	22003	significant data lost due to truncation
15	errDATETIMEOVF	22008	date/time value overflow
160	errDBEXISTS	42000	database already exists
41	errDBNOTOPEN	42000	database not open
177	errDBOPEN	42000	database has already been opened
169	errDBOPENMODE	42000	another database is already open in different mode
40	errDBTBO	42000	databases to be opened already specified
176	errDBUNAVAIL	42000	database unavailable due to exclusive access rules
69	errDDLNOTEXEC	42000	prior prepared DDL statement not executed
159	errDEFEREDBLOB	42000	cannot refer to blob column in WHERE in deferred reading mode
115	errDESCRANGE	HY091	invalid descriptor field identifier
182	errDIFFCONNS	42000	statements from different connections
101	errDISTINCT	42000	only one DISTINCT aggregate function is allowed
67	errDIVBY0	22012	division by zero
58	errDRVMEMORY	HY001	driver memory allocation error
10	errDUPCOLNAME	42S21	column already declared in table
12	errDUPCURSOR	3C000	duplicate cursor name
94	errDUPJOINCOL	42000	duplicate join column
90	errDUPLICATE	42000	duplicate primary/unique key value
71	errDUPPROC	42000	duplicate stored procedure name

11	errDUPTABNAME	42S21	table with the same name already been created
68	errESCAPE	42000	escape clause syntax error
64	errESCAPECHAR	22019	invalid escape character
138	errEXCLUSIVE	RX020	operation requires exclusive database access
97	errFCNARG	21000	incorrect number of function arguments
146	errFCNRANGE	HY095	function type out of range
35	errFCNSEQ	HY010	function call sequence error
49	errFETCHTYPE	HY106	fetch type out of range
26	errFILEIO	RX006	file I/O error
8	errFKPKMISMATCH	42000	foreign/primary key columns do not match
88	errFKSETNULL	42000	SET NULL cannot be specified when nulls are not allowed
5	errGENDDLFILE	RX004	unable to open RDMe core-level DDL file
17	errGENFILE	RX005	unable to open file
103	errGROUPBYAGGFN	42000	aggregate functions not allowed in GROUP BY
77	errHDBCINUSE	08002	connection already in use
125	errIEF	RX016	import/export error
25	errINITDB	42000	unable to initialize database
89	errINSERTCOLREF	42000	invalid column reference in INSERT expression
42	errINSERTVALS	21S01	insert value list does not match column list
28	errINVARG	HY009	invalid argument value
181	errINVARGTYPE	HY009	invalid argument type
121	errINVCONVERT	HYC00	invalid conversion
135	errINVDATEFMT	42000	invalid date format
136	errINVDATESEP	42000	invalid date separator
107	errINVDESCIDX	07009	invalid descriptor index
112	errINVIRDMOD	HY016	cannot modify an implementation row descriptor
29	errINVNULL	HY009	invalid use of null pointer
137	errINVOPENMODE	42000	invalid db open mode
110	errINVOPNOW	HY011	invalid operation at this time
70	errINVPARAM	42000	invalid use of parameter marker
65	errINVSTATE	RX008	invalid statement state
100	errINVSTMT	RX015	invalid statement type
124	errINVTFSSPEC	42000	invalid TFS location spec, should be: "@hos-tname:port"
174	errINVTRID	RX027	invalid transaction id
111	errINVTXTYPE	HY012	invalid transaction operation code
127	errJNIERROR	RX018	JNI system error
74	errJOINCOLS	42000	joined columns must match exactly
75	errKEYCOLLIMIT	42000	too many columns declared in foreign/rpimary key
162	errLOCKMODE	RX026	illegal locking mode
132	errLOCKSACTIVE	25000	operation not allowed due to active read locks

173	errLOCKSFREED	25000	cursor's read locks freed by intervening commit/rollback
158	errMAXROWS	42000	maxrows can only be specified with CREATE CIRCULAR TABLE
134	errMULTIDBTRANS	25000	multiple database transactions are not allowed
170	errNESTEDAGGS	42000	cannot call an aggregate function within an aggregate function
180	errNOBLOBGROUP	42000	blob columns cannot be referenced in a SELECT with GROUP BY
179	errNOBLOBSORT	42000	sorting on a blob column is not allowed
92	errNOCASCADE	42000	ON UPDATE CASCADE not allowed when foreign key column is used in a key
44	errNOCOLUMN	42S22	column not found
4	errNOCRDB	42000	no CREATE DATABASE has been issued
23	errNODB	42000	unable to open database
13	errNODOMAIN	42000	specified domain name not found
122	errNOINDVAR	22002	indicator variable required but not supplied
95	errNOJOINCOLMATCH	42000	no matching join columns
18	errNOMEMORY	HY013	memory management error
133	errNOPENDBS	42000	DDL requires that no databases be open
156	errNOPGSANDROWS	42000	cannot specify both MAXPGS and MAXROWS options
72	errNOPROC	RX009	stored procedure file not found
91	errNOSETNULL	42000	SET NULL not allowed with ON UPDATE
119	errNOTCAPABLE	HYC00	driver not capable
106	errNOTCURSOR	07005	prepared statement is not a valid cursor
120	errNOTIMPLEMENTED	HYC00	optional feature not implemented
166	errNOTINPROC	42000	positioned UPDATE/DELETE not allowed in stored procedure
131	errNOTLOCKED	25000	table is not locked
46	errNOTNULL	42000	must specify value for column
27	errNOTOPEN	08003	connection is not open
109	errNOTPREPARED	HY007	associated statement is not prepared
82	errNOTREGISTERED	42000	reference to unregistered UDF/UDP/Virtual Table
53	errNOTSELECT	24000	current statement is not SELECT
123	errNOTSUPPORTED	IM001	function not supported
139	errNOTUPDATEABLE	42000	SELECT or column is not updateable
163	errNOUPDCOLS	42000	no columns have been updated
187	errNOWHEREBLOB	42000	blob columns cannot be used in WHERE clause of a SELECT with ORDER BY
149	errNULLRANGE	HY099	nullable type out of range
73	errNUMARGS	21000	invalid number of arguments specified
98	errNUMFCNARGS	42000	invalid number of specified function arguments

45	errNUMPAR	07002	insufficient number of parameters specified
30	errOPENHDBCS	HY010	must free all connection handles first
62	errOPTCHANGED	01S02	option value changed to default
60	errOPTIONRANGE	HY092	invalid attribute/option identifier
183	errOUTERJOIN	42000	unable to process outer join specification
117	errPARMTYPE	HY105	invalid parameter type
34	errPARNUMBER	07009	invalid descriptor index (parameter number)
7	errPKNOTFOUND	42000	matching primary key does not exist in referenced table
2	errPSPFAILURE	RX003	psp subsystem initialization failure
185	errQUERYUDF	42000	query() cannot be used in SELECT result column
54	errRANGE	22003	numeric value out of range
38	errRDMERROR	RX007	RDMe runtime error
178	errRDONLYDB	42000	database is opened for read only
167	errRDONLYFLAG	25000	Inconsistent read-only transaction commit/rollback/end call
99	errREADONLY	RX014	operation is read only
56	errREFINTEGRITY	23000	referential integrity error
78	errREGISTERED	42000	UDF/UDP/XTF already registered
93	errRESTRICTED	42000	changes to referenced restricted primary/unique key not allowed
129	errROTRACT	25000	read-only transaction is active
61	errROWERROR	01S01	error in row
50	errROWVALUE	HY107	row value out of range
116	errSCALE	HY094	invalid scale value
148	errSCOPERANGE	HY098	scope out of range
43	errSELECTCOLS	21S02	select result columns do not match column list
184	errSHOWPLAN	42000	invalid access--use rsqlShowPlan/SQLShowPlan function
96	errSORTCOLREF	42000	invalid order/group by column reference
152	errSQLTYPE	HY004	invalid SQL data type
3	errSYNTAX	42000	syntax error
189	errSYSTEM	RX999	system error
168	errTABLEREF	42000	duplicate table reference in FROM clause
14	errTABNOTFOUND	42S02	table not found
47	errTABNOTLISTED	42000	table name not in FROM clause
126	errTFSERROR	RX017	TFS system error
154	errTFSFAILURE	RX024	unable to connect to TFS
161	errTFSINIT	RX025	TFS already initialized
118	errTIMEOUT	HYT00	timeout expired
22	errTOOLONG	22001	string literal too long
36	errTRACTIVE	25000	transaction is active

37	errTRNOTACT	25000	transaction not active
1	errTRUNCATE	01004	data truncation
108	errTXUNKNOWN	25S01	transaction state unknown
32	errTYPEATTR	07006	data type attribute violation
63	errTYPEMISMATCH	42000	data type mismatch
83	errUDF	RX011	user-defined function error
86	errUDFARG	21000	invalid funtion argument type
87	errUDFNOARGS	21000	incorrect number of funtion arguments
85	errUDFNORESET	RX013	no reset callback for aggregate user-defined fun
84	errUDFNOVAL	RX012	no result from user-defined function
175	errUNIONOPEN	42000	db union open invalid when other database is open
150	errUNIQUERANGE	HY100	Uniqueness option type out of range
130	errUNLOCKINTRANS	25000	unlock not allowed in a transaction
155	errVARCHARLEN	42000	must specify '(length)' with variable size columns
80	errVIRTABCOLTYPE	42000	data type not allowed for virtual table columns
81	errVIRTABERROR	RX010	virtual table function error
79	errVIRTABPOS	42000	all standard tables must be declared before first virtual table
186	errVTACCESS	42000	virtual table access restricted to INSERT or SELECT
66	errWHERECALCS	42000	aggregate functions not allowed in WHERE

Table 33. RDMe SQL API Error Codes by Value

Error Code	Enum Identifier	SQL State	Description
1	errTRUNCATE	01004	data truncation
2	errPSPFAILURE	RX003	psp subsystem initialization failure
3	errSYNTAX	42000	syntax error
4	errNOCRDB	42000	no CREATE DATABASE has been issued
5	errGENDDLFILE	RX004	unable to open RDMe core-level DDL file
6	errBADREFDCOL	42000	column is not declared in referenced table
7	errPKNOTFOUND	42000	matching primary key does not exist in referenced table
8	errFKPKMISMATCH	42000	foreign/primary key columns do not match
9	errBADFORKEYCOL	42000	foreign key column is not declared in table
10	errDUPCOLNAME	42S21	column already declared in table
11	errDUPTABNAME	42S21	table with the same name already been created
12	errDUPCURSOR	3C000	duplicate cursor name
13	errNODOMAIN	42000	specified domain name not found
14	errTABNOTFOUND	42S02	table not found
15	errDATETIMEOVF	22008	date/time value overflow
16	errBADKEYCOL	42000	key column not found
17	errGENFILE	RX005	unable to open file
18	errNOMEMORY	HY013	memory management error

19	errBADFORMAT	42000	bad formatting specification
20	errBADBINLITERAL	22005	bad binary literal specification
21	errBADLITERAL	22005	bad literal specification
22	errTOOLONG	22001	string literal too long
23	errNODB	42000	unable to open database
24	errCATNOTFOUND	42000	unable to open catalog file
25	errINITDB	42000	unable to initialize database
26	errFILEIO	RX006	file I/O error
27	errNOTOPEN	08003	connection is not open
28	errINVARG	HY009	invalid argument value
29	errINVNULL	HY009	invalid use of null pointer
30	errOPENHDBCS	HY010	must free all connection handles first
31	errBADDATALEN	HY090	invalid string or buffer length
32	errTYPEATTR	07006	data type attribute violation
33	errCOLNUMBER	07009	invalid descriptor index (column number)
34	errPARNUMBER	07009	invalid descriptor index (parameter number)
35	errFCNSEQ	HY010	function call sequence error
36	errTRACTIVE	25000	transaction is active
37	errTRNOTACT	25000	transaction not active
38	errRDMERROR	RX007	RDMe runtime error
39	errCONNECTED	08000	must call before connect
40	errDBTBO	42000	databases to be opened already specified
41	errDBNOTOPEN	42000	database not open
42	errINSERTVALS	21S01	insert value list does not match column list
43	errSELECTCOLS	21S02	select result columns do not match column list
44	errNOCOLUMN	42S22	column not found
45	errNUMPAR	07002	insufficient number of parameters specified
46	errNOTNULL	42000	must specify value for column
47	errTABNOTLISTED	42000	table name not in FROM clause
48	errCOLNOTFOUND	42S22	column name not found
49	errFETCHTYPE	HY106	fetch type out of range
50	errROWVALUE	HY107	row value out of range
51	errCURPOS	HY109	invalid cursor position
52	errCURSTATE	24000	invalid cursor state
53	errNOTSELECT	24000	current statement is not SELECT
54	errRANGE	22003	numeric value out of range
55	errDATALOST	22003	significant data lost due to truncation
56	errREFINTEGRITY	23000	referential integrity error
57	errCLOSEHDBC	HY010	connection not closed
58	errDRVMEMORY	HY001	driver memory allocation error
59	errATTRVALUE	HY024	invalid attribute value
60	errOPTIONRANGE	HY092	invalid attribute/option identifier

61	errROWERROR	01S01	error in row
62	errOPTCHANGED	01S02	option value changed to default
63	errTYPEMISMATCH	42000	data type mismatch
64	errESCAPECHAR	22019	invalid escape character
65	errINVSTATE	RX008	invalid statement state
66	errWHERECALCS	42000	aggregate functions not allowed in WHERE
67	errDIVBY0	22012	division by zero
68	errESCAPE	42000	escape clause syntax error
69	errDDLNOTEXEC	42000	prior prepared DDL statement not executed
70	errINVPARAM	42000	invalid use of parameter marker
71	errDUPPROC	42000	duplicate stored procedure name
72	errNOPROC	RX009	stored procedure file not found
73	errNUMARGS	21000	invalid number of arguments specified
74	errJOINCOLS	42000	joined columns must match exactly
75	errKEYCOLLIMIT	42000	too many columns declared in foreign/rpimary key
76	errBADOUTERJOIN	42000	no access path between outer joined tables
77	errHDBCINUSE	08002	connection already in use
78	errREGISTERED	42000	UDF/UDP/XTF already registered
79	errVIRTABPOS	42000	all standard tables must be declared before first virtual table
80	errVIRTABCOLTYPE	42000	data type not allowed for virtual table columns
81	errVIRTABERROR	RX010	virtual table function error
82	errNOTREGISTERED	42000	reference to unregistered UDF/UDP/Virtual Table
83	errUDF	RX011	user-defined function error
84	errUDFNOVAL	RX012	no result from user-defined function
85	errUDFNORESET	RX013	no reset callback for aggregate user-defined fun
86	errUDFARG	21000	invalid funtion argument type
87	errUDFNOARGS	21000	incorrect number of funtion arguments
88	errFKSETNULL	42000	SET NULL cannot be specified when nulls are not allowed
89	errINSERTCOLREF	42000	invalid column reference in INSERT expression
90	errDUPLICATE	42000	duplicate primary/unique key value
91	errNOSETNULL	42000	SET NULL not allowed with ON UPDATE
92	errNOCASCADE	42000	ON UPDATE CASCADE not allowed when foreign key column is used in a key
93	errRESTRICTED	42000	changes to referenced restricted primary/unique key not allowed
94	errDUPJOINCOL	42000	duplicate join column
95	errNOJOINCOLMATCH	42000	no matching join columns
96	errSORTCOLREF	42000	invalid order/group by column reference
97	errFCNARG	21000	incorrect number of function arguments
98	errNUMFCNARGS	42000	invalid number of specified function arguments

99	errREADONLY	RX014	operation is read only
100	errINVSTMT	RX015	invalid statement type
101	errDISTINCT	42000	only one DISTINCT aggregate function is allowed
102	errCANCELED	HY008	statement execution canceled by user
103	errGROUPBYAGGFN	42000	aggregate functions not allowed in GROUP BY
104	errBADCTYPE	07006	invalid C data type
105	errBADSTYPE	07006	invalid SQL data type
106	errNOTCURSOR	07005	prepared statement is not a valid cursor
107	errINVDESCIDX	07009	invalid descriptor index
108	errTXUNKNOWN	25S01	transaction state unknown
109	errNOTPREPARED	HY007	associated statement is not prepared
110	errINVOPNOW	HY011	invalid operation at this time
111	errINVTXTYPE	HY012	invalid transaction operation code
112	errINVIRDMOD	HY016	cannot modify an implementation row descriptor
113	errAUTODESCR	HY017	invalid use of implicit descriptor handle
114	errBADDESCINFO	HY021	inconsistent descriptor information
115	errDESCRANGE	HY091	invalid descriptor field identifier
116	errSCALE	HY094	invalid scale value
117	errPARMTYPE	HY105	invalid parameter type
118	errTIMEOUT	HYT00	timeout expired
119	errNOTCAPABLE	HYC00	driver not capable
120	errNOTIMPLEMENTED	HYC00	optional feature not implemented
121	errINVCONVERT	HYC00	invalid conversion
122	errNOINDVAR	22002	indicator variable required but not supplied
123	errNOTSUPPORTED	IM001	function not supported
124	errINVTFSSPEC	42000	invalid TFS location spec, should be: "@hos-tname:port"
125	errIEF	RX016	import/export error
126	errTFSERROR	RX017	TFS system error
127	errJNIERROR	RX018	JNI system error
128	errCOMERROR	RX019	RPC communication error
129	errROTRACT	25000	read-only transaction is active
130	errUNLOCKINTRANS	25000	unlock not allowed in a transaction
131	errNOTLOCKED	25000	table is not locked
132	errLOCKSACTIVE	25000	operation not allowed due to active read locks
133	errNOPENDBS	42000	DDL requires that no databases be open
134	errMULTIDBTRANS	25000	multiple database transactions are not allowed
135	errINVDATEFMT	42000	invalid date format
136	errINVDATESEP	42000	invalid date separator
137	errINVOPENMODE	42000	invalid db open mode
138	errEXCLUSIVE	RX020	operation requires exclusive database access
139	errNOTUPDATEABLE	42000	SELECT or column is not updateable

140	errBLOBDEFAULT	42000	default values not allowed on long var{char|binary} columns
141	errBLOBEXPR	42000	blobs cannot be referenced in expressions in deferred mode
142	errDATAATEXEC	RX021	data-at-exec params only allowed with INSERT VALUES/UPDATE
143	errBLOBPARSONLY	RX022	data-at-exec params only allowed for blob (long var...) columns
144	errBADBLOBPAR	RX023	data-at-exec param type not compatible with blob (long var...) column
145	errCONCATNULL	HY020	attempt to concatenate a null value
146	errFCNRANGE	HY095	function type out of range
147	errCOLRANGE	HY097	column type out of range
148	errSCOPERANGE	HY098	scope out of range
149	errNULLRANGE	HY099	nullable type out of range
150	errUNIQUERANGE	HY100	Uniqueness option type out of range
151	errACCURANGE	HY101	Accuracy option type out of range
152	errSQLTYPE	HY004	invalid SQL data type
153	errCIRCTABREF	42000	circular tables cannot be referenced
154	errTFSFAILURE	RX024	unable to connect to TFS
155	errVARCHARLEN	42000	must specify '(length)' with variable size columns
156	errNOPGSANDROWS	42000	cannot specify both MAXPGS and MAXROWS options
157	errCIRTABDELETE	42000	cannot delete rows from a circular table
158	errMAXROWS	42000	maxrows can only be specified with CREATE CIRCULAR TABLE
159	errDEFEREDBLOB	42000	cannot refer to blob column in WHERE in deferred reading mode
160	errDBEXISTS	42000	database already exists
161	errTFSINIT	RX025	TFS already initialized
162	errLOCKMODE	RX026	illegal locking mode
163	errNOUPDCOLS	42000	no columns have been updated
164	errAUTOCOMMIT	42000	operation not allowed when autocommit is enabled
165	errCURSORTABLE	42000	positioned UPDATE/DELETE table does not match cursor's
166	errNOTINPROC	42000	positioned UPDATE/DELETE not allowed in stored procedure
167	errRDONLYFLAG	25000	Inconsistent read-only transaction commit/rollback/end call
168	errTABLEREF	42000	duplicate table reference in FROM clause
169	errDBOPENMODE	42000	another database is already open in different mode
170	errNESTEDAGGS	42000	cannot call an aggregate function within an aggre-

			gate function
171	errCHARREQD	42000	char/wchar type is required
172	errCONNECT	08001	unable to connect
173	errLOCKSFREED	25000	cursor's read locks freed by intervening commit/rollback
174	errINVTRID	RX027	invalid transaction id
175	errUNIONOPEN	42000	db union open invalid when other database is open
176	errDBUNAVAIL	42000	database unavailable due to exclusive access rules
177	errDBOPEN	42000	database has already been opened
178	errRDONLYDB	42000	database is opened for read only
179	errNOBLOBSORT	42000	sorting on a blob column is not allowed
180	errNOBLOBGROUP	42000	blob columns cannot be referenced in a SELECT with GROUP BY
181	errINVARGTYPE	HY009	invalid argument type
182	errDIFFCONNS	42000	statements from different connections
183	errOUTERJOIN	42000	unable to process outer join specification
184	errSHOWPLAN	42000	invalid access--use rsqlShowPlan/SQLShowPlan function
185	errQUERYUDF	42000	query() cannot be used in SELECT result column
186	errVTACCESS	42000	virtual table access restricted to INSERT or SELECT
187	errNOWHEREBLOB	42000	blob columns cannot be used in WHERE clause of a SELECT with ORDER BY
189	errSYSTEM	RX999	system error