Contents

Overview

About SQL-Sombrero/VBX for DB-Library

Installation

What You Will Need: Environment Information

Instructions: How to Install

Files Created and Affected by Install

Application Architecture

Guidelines for your Application

Handling Errors and Messages

Initializing the SQL-Sombrero/VBX Library

Establishing a SQL Server Connection

Communicate Transact-SQL statements to the SQL Server

Obtain and Process Results

Close Connections to the SQL Server

Function Descriptions

CompileDate

SombreroVersion

SqlADData

SqlADLen

SqlAltColId

SqlAltLen

SqlAltOp

SqlAltType

SqlAltUType

SqlBCPBatchPC

SqlBCPColfmt

SqlBCPColumnFormat

SqlBCPColumns

SqlBCPColumnsPC

SqlBCPControl

SqlBCPDonePC

SqlBCPExec

SqlBCPInit
SqlBCPInitPC
SqlBCPReadFmt
SqlBCPSendRowPC
SqlBCPSetDataPC
SqlBCPSetL
SqlBCPWriteFmt
SqlByList
SqlCancel
SqlCanQuery
SqlChange
SqlClose
SqlClrBuf
SqlClrOpt
SqlCmd
SqlCmdRow
SqlColBrowse
SqlColLen
SqlColName
SqlColSource
SqlColType
SqlColUType
SqlCount
SqlCurCmd
SqlCurRow
SqlData
SqlDataReady
SqlDateCrack
SqlDatLen
SqlDead
SqlExec
SqlExit
SqlFirstRow
SqlFreeBuf
SqlFreeLogin
SqlGetChar
SqlGetOff

SqlGetRow
SqlGetTime
SqlHasRetStat
SqlInit
SqlIsAvail
SqlIsCount
SqlIsOpt
SqlLastRow
SqlLogin
SqlMoreCmds
SqlMoreText
SqlName
SqlNextRow
SqlNumAlts
SqlNumCols
SqlNumCompute
SqlNumOrders
SqlNumRets
SqlOk
SqlOpen
SqlOpenConnection
SqlOrderCol
SqlPrType
SqlQual
SqlResults
SqlRetData
SqlRetLen
SqlRetName
SqlRetStatus
SqlRetType
SqlRows
SqlRowType
SqlRpcInit
SqlRpcParam
SqlRpcSend
SqlRPwClr
SqlRPwSet

SqlSend
SqlSendCmd
SqlServerEnum
SqlSetAvail
SqlSetLApp
SqlSetLHost
SqlSetLNatLang
SqlSetLoginTime
SqlSetLPwd
SqlSetLUser
SqlSetMaxProcs
SqlSetOpt
SqlSetTime
SqlStrCpy
SqlStrLen
SqlTabBrowse
SqlTabCount
SqlTabName
SqlTabSource
SqlTsNewLen
SqlTsNewVal
SqlTsPut
SqlTsUpdate
SqlTxPtr
SqlTxTimeStamp
SqlTxTsNewVal
SqlTxTsPut
SqlUse
SqlWinExit
SqlWriteText

About SQL-Sombrero/VBX for DB-Library

The SQL-Sombrero/VBX is a library of functions for use with Sybase and Microsoft SQL Server products. The functions are used to build SQL Server front-end applications for the Windows operating environment. The SQL-Sombrero/VBX is an interface to DB-Library, a SQL Server Client-Library API for C programmers.

SQL-Sombrero/VBX allows Visual Basic to interact directly with SQL Server. The functions allow a developer to easily make connections with SQL Server databases, send Transact-SQL statements, and process their results.

What You Will Need: Environment Information

To Use SQL-Sombrero/VBX you will need the following:

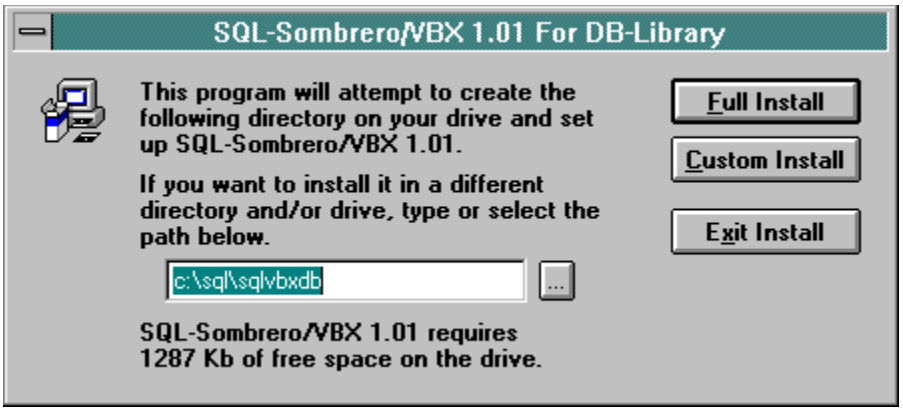
- Microsoft Windows release 3.1 or later.
- 80386 (20MHz) or higher processor.
- 4 megabytes of RAM.
- 2 megabytes of free disk space for software and environment.
- Microsoft or SYBASE SQL Server Open Client software (runtime DB-Library and Net-Library) for Windows and compatible network.

Instructions: How to Install

SQL-Sombrero/VBX comes with one install diskette.

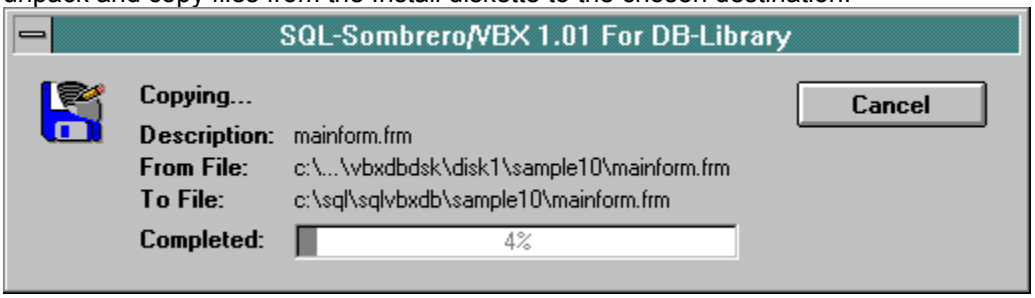
Insert the Install diskette and run INSTALL.EXE from within windows.

The default is a directory called C:\SQL\SQLVBXDB. A drop down list box facilitates choosing an alternative destination.



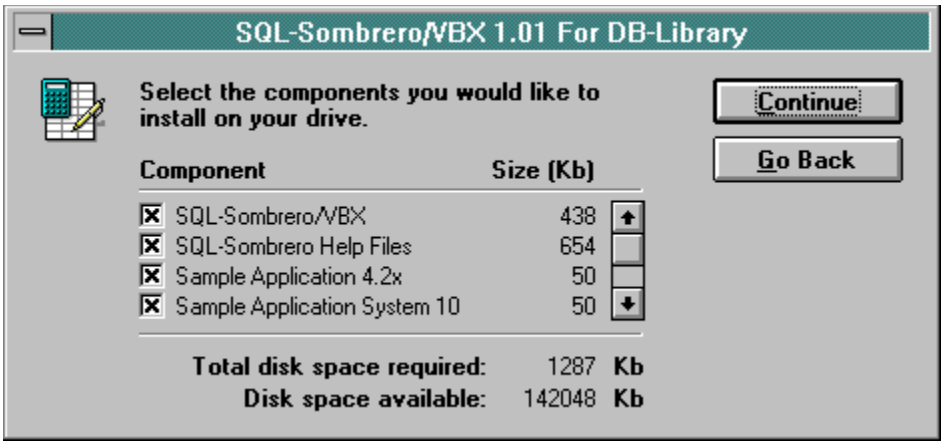
Full Install

Press the Full Install button to install SQL-Sombrero/VBX completely. The installation process begins to unpack and copy files from the Install diskette to the chosen destination.



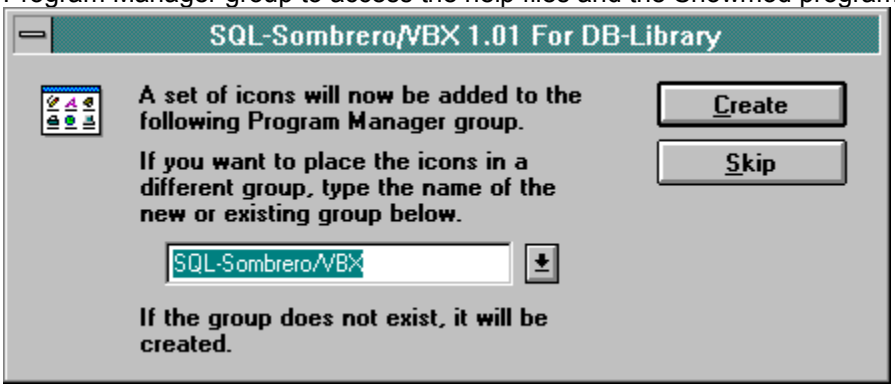
Custom

The Custom Install procedure allows individual components of the SQL-Sombrero/VBX package to be installed separately.

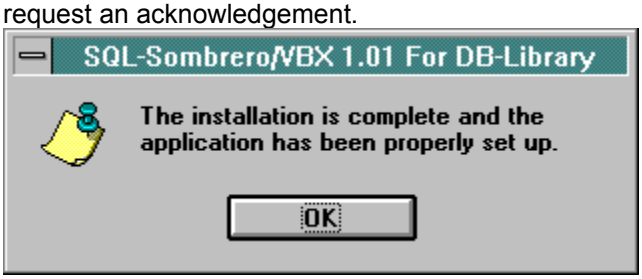


Note: it is important that the latest DLLs are used in order to achieve maximum functionality.

When the installation of files is complete, the install procedure will ask you if you wish to create a Program Manager group to access the help files and the Showmod program.



After creating the Program Manager group the install will inform you that the installation is complete and request an acknowledgement.



Files Created and Affected by Install

If you choose the Custom installation option there are a total of five (5) different components that can be installed.

1. SQL-Sombrero/VBX

- this component consists of the VBX file with its license file and the file of declarations for the SQL-Sombrero/VBX functions.

If this component is selected then the following files are installed. The files are installed into the directory choosen in the installation procedure as described above.

SQLVBXDB.VBX

SQLNDEF.BAS

SQLVBXDB.LIC

README.TXT

INSTALL.LOG (Shows files installed and their locations)

2. SQL-Sombrero/VBX Help File

- this component consists of the Help file for SQL-Sombrero/VBX as well as other files to allow the navigator option to work with the help file.

If this component is selected then the following files are installed. The files are installed into the directories as shown below.

SQLVBXDB.HLP	- default directory
SQLVBXDB.DHN	- default directory
D2HNAV.EXE	- windows directory
D2HNAV.HLP	- windows directory
MSOUTLIN.VBX	- windows system directory
D2HLINK.DLL	- windows system directory

3. Sample Application 4.2x

- this is a sample application using the SQL-Sombrero/VBX for SQL Servers 4.2x. This sample requires the PUBS database to be installed.

LOGON.FRM	- default/sample42 directory
MAINFORM.FRM	- default/sample42 directory
GBASS.BAS	- default/sample42 directory
SAMPLE42.MAK	- default/sample42 directory
GLOBAL.BAS	- default/sample42 directory

4. Sample Application System 10

- this is a sample application using the SQL-Sombrero/VBX for SYBASE SQL Server System 10.

This sample requires the PUBS2 database to be installed.

LOGON.FRM - default/sample10 directory

MAINFORM.FRM - default/sample10 directory

GBASS.BAS - default/sample10 directory

SAMPLE10.MAK - default/sample10 directory

GLOBAL.BAS - default/sample10 directory

5. Show Active Modules

- this is a program which will scan all module currently running in the Windows environment and show details of those modules.

SHOWMOD.EXE - default directory

The file SQLVBXDB.LIC is the licence file. This file must reside in the same directory as the file SQLVBXDB.VBX. This file is required if the SQL-Sombrero/VBX for DB-Library is to be used in developing an application. The SQLVBXDB.LIC file is not required for an application that is distributed as an EXE file. Sample42 is for the Pubs database and Sample10 is for the Pubs2 database.

Please refer to the inside of this manual's front cover for more information regarding the LIC file.

Guidelines for your Application

The following is intended as a set of guidlelines for constructing an application based on SQL-Sombrero/VBX.

1. Construct the error and message handlers
2. Initialize the SQL-Sombero/VBX control.
3. Establish a connection with the SQL Server.
4. Communicate Transact-SQL statements to the SQL Server.
5. Obtain and process results.
6. Close connections to the SQL Server and terminate the application.

This section will elaborate on each of these architectural guidelines. Excerpts from the sample application provided with the installation have been included to help with the explanation.

Handling Errors and Messages

Errors detected by the SQL-Sombrero/VBX trigger the error handler and the message handler. The error handler and the message handler must be coded to take appropriate action when errors occur and messages are returned.

To incorporate the error and message handling procedures in your application perform the following.

- Choose add file from the File menu, to add the file SQLVBXDB.VBX to your project. Visual Basic will add a custom control, shown as a SQL box, to the bottom of the Toolbox.

- Select the custom control and place it on your main form. Only one copy should be within your application. The main form should now include the object SQLVBXDB1 which includes procedure templates for the event handlers: SQLVBXDB1_Error and SQLVBXDB1_Message.

The following sample application excerpt shows a simple example of using the error handler.

```
Sub SQLVBXDB1_ERROR (Sqlconn As Integer, Severity As Integer, ErrorNum
As Integer, OsError As Integer, errorstr As String, OsErrorStr As
String, RetCode As Integer)
'   This is a SQL Server callback routine
'
' When the server needs to inform the user of a error this callback is
called.For
' example if the user tries to logon to a server which does not exist
then the error
' message string (errorstr) will contain text indicating that a
connection with the
' server cannot be made. This routine is also called when syntax errors
are discovered
'in SQL commands submitted for execution.
    MsgBox errorstr
End Sub
SQLVBXDB1_MESSAGE is the built in message handler which must be coded to
handle messages from SQL Server.  The Message handler is triggered
automatically when the server sends a message back to the front end.
Sub SQLVBXDB1_MESSAGE (Sqlconn As Integer, message As Long, State As
Integer, Severity As Integer, msgstr As String, ServerName As String,
ProcName As String, LineNum As Integer)
'This is a SQL Server callback routine
'
'When the server needs to inform the user of a status change this
callback is used.
' For example when the user changes databases using the SqlUse function
this event
' procedure will be called and the message string (msgstr) will contain
text
' indicating the new database name. The variable severity can be used to
filter the
```

```
' messages so that only messages which need to be seen can be displayed
to the user
    MsgBox msgstr
End Sub
```

Initializing the SQL-Sombrero/VBX Library

In order to avoid conflicts with other applications, your application must register itself with the SQL-Sombrero/VBX Library. This is done through a call to **SqlInit**. Before calling any other SQL-Sombrero/VBX function a call must be made to **SqlInit**.

The following sample application excerpt shows using SqlInit in a Logon button. Please refer to Logon.Frm provided with this installation.

```
Sub logonbut_Click ()
    dblib = SqlInit()
```

Establishing a SQL Server Connection

After initializing the application, one or more SQL Server connections may be opened. Your application will use a connection to communicate with the SQL Server.

To open a connection and login to the SQL Server use the **SQLOpenConnection** function. The sample application shows the following example of how to open a connection.

```
'   Once the information needed to open a connection is
'   obtained from the user open a connection with the server
'   using the SqlOpenConnection function.  This function
'   returns a connection pointer if the connection is made.
'   if no connection is made then the connection pointer will
'   be zero (NULL)
    dbconn = SqlOpenConnection(sid, usid, pword, "", "SombreroApp1")
'   If the connection is made the next thing we need to do
'   is point to the correct database which has the data
'   required for the application.  In this case the data is
'   in the authors table found in the pubs database for SQL
'   Server 4.2 or in the pubs2 database for SYBASE System 10
    If dbconn <> 0 Then
        ret = SqlUse(dbconn, datbase)
        If ret <> 1 Then
            SqlClose (dbconn)
            dbconn = 0
        End If
    Else
        Exit Sub
    End If
```

Above the connection is tested, if it's non-zero the function **SQLUse** is used to change the current database.

Communicate Transact-SQL statements to the SQL Server

The functions **SQLCmd** and **SQLExec** are used to send Transact-SQL statements to the SQL Server.

The **SQLCmd** function places command in the command buffer, and **SQLExec** causes them to be executed. The following sample application code fragment illustrates their usage.

```
'   Once the user has logged on to the server the list box will be
'   populated with a list of the Author Id, Last Name, and First Name
'
'   The first function is to place the SQL Statement required to get
'   this information from the server into the command buffer using
'   the SqlCmd function
 ret = SqlCmd(dbconn, "select 'Author Id' = au_id , 'Last Name' =
au_lname , 'First Name' = au_fname  from authors")
    If ret = 1 Then
'   The SqlExec function is then used to send the SQL Statement to the
'   server for execution.  It is at this point that syntax checking is
'   performed
   ret = SqlExec(dbconn)
   If ret = 1 Then
```

A return of 1 indicates success, the program is now ready to process results.

Obtain and Process Results

Many functions exist to process results. Here the functions **SQLResults**, **SQLNumCols**, **SQLColName**, **SQLNextRow** and **SQLData** are highlighted to show a typical example.

```
' If the SqlCmd function had been passed more than one SQL command then
you must
' perform a SqlResults for each result set being sent back.  The end of
result sets
' will be indicated by a NOMORERESULTS(2) return from SqlResults
      ret = SqlResults(dbconn)
      If ret = 1 Then
' Once the SqlResults returns with the indication that a result set is
available we
' can get the number of columns in the result set.  In this application
this function
' is not required since we know how many columns were requested.  If the
application
' allowed for AdHoc SQL requests then this function is used to indicate
the number of
' columns of data available.
     cols% = SqlNumCols(dbconn)
' To get the column headings we use the SqlColName function. This
function will return
' either the column name based on the internal column name or if the
syntax 'Column
' Name' = colname is used to override the internal column name.  In our
example we
' have chosen to override the column name
     For c% = 1 To cols%
         colnam$ = SqlColName(dbconn, c%)
     Next
' Once the result set is available for processing each row needs to be
retrieved.
' This is accomplished by calling SqlNextRow until the function  returns
NOMOREROWS
' (-2)
     ret = SqlNextRow(dbconn)
     While ret <> NOMOREROWS
' For each column in the result set we call the function SqlData to get
the data
' returned for the column.  The data returned is a string representation
of the data
' in the result column.  The data is right trimmed when returned.  The
data when
' return is being concatonated with tabs (chr(9)) between each item to
```

```
populate the
' drop down list box with three columns


          For c% = 1 To cols%
         t(c%) = SqlData(dbconn, c%)
          Next
          aitem$ = t(1) & Space(15 - Len(t(1))) & Left(t(2) &
                                                Space(41 - Len(t(2))), 20)
& t(3)
          author_list.AddItem aitem$
          ret = SqlNextRow(dbconn)
      Wend
       End If
   End If
    End If
    If ret <> NOMOREROWS Then
   Exit Sub
    End If
```

Close Connections to the SQL Server

To close a specific connection use **SQLClose**. To close all connections use **SQLExit**.

The following fragment, extracted from the sample application provided, shows an example of exit button code.

```
Sub exitbut_Click ()
    If dbconn <> 0 Then
   SqlClose (dbconn)
    End If
    sqlwinexit
    sqlexit
    End
End Sub
```

CompileDate

Returns the compile date of SQL-Sombrero/VBX into a string

Syntax

cdate$=CompileDate()

Remarks

This function can be called at any time.

SombreroVersion

Returns the version of SQL-Sombrero into a string

Syntax

ver$=SombreroVersion()

Remarks

This function can be called at any time.

SqlAData

Retrieves a string representation of the column data for a compute clause column.

Syntax

SqlAData (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

The COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets. The computeid% is returned by SqlNextRow.

column%

The column number. The first column number returned is 1.

Results

The return value is a string containing the data for a particular compute column.

Example

```
'Place commands into the command buffer.
cmd$ = "SELECT cust_name(custid), custid, amount FROM cust_amount"
cmd$ = cmd$ + " ORDER BY  custid"
cmd$ = cmd$ + " COMPUTE SUM(amount) BY custid"
i_ret% = SqlCmd(connectid%, cmd$)
'Send commands to the SQL Server and execute.
i_ret% = SqlExec(connectid%)
i_ret% = SqlResults(connectid%)
'Check out the  COMPUTE clause results.
IF i_ret% = SUCCEED THEN
   DO UNTIL i_ret% = NOMOREROWS
      i_ret% = SqlNextRow%(connectid%)
      IF i_ret% = NOMOREROWS THEN Exit DO
      IF i_ret% = REGROW THEN
         PRINT "regular row was returned."
         PRINT
      ELSE
         'This row is a COMPUTE clause result
         'and i_ret% is the computeid% of this
         'COMPUTE clause
```

```
        amt$ = SqlAData(connectid%, i_ret%, 1)
        PRINT "cust amt =  " + amt$
        PRINT
    END IF
LOOP
```

Remarks

If the value in the column is NULL then the string "NULL" is returned.

After each call to SqlNextRow that returns a value greater than 0, use SqlAData to obtain the data in a particular COMPUTE clause column. The data is not null-terminated.

See Also

SqlADLen, SqlAltLen, SqlAltType, SqlNextRow, SqlNumAlts

SqlADLen

Returns the actual length of the data for a compute column in bytes.

Syntax

SqlADLen (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets. This is obtained from the SqlNextRow function.

column%

The column number. The first column is number 1.

Results

The length, in number of bytes, of the data in a COMPUTE clause column. -1 is returned when no such column or COMPUTE clause exists. 0 is returned when the data has a null value. You can get the data for the column using SqlAData.

Example

```
'Place the command into the command buffer.
cmd$ = "SELECT cust_name FROM customer"
cmd$ = cmd$ + " ORDER BY  cust_name"
cmd$ = cmd$ + " COMPUTE MAX(cust_name)"
'Send the command to SQL Server and execute.
i_ret% = SqlCmd(connectid%, cmd$)
i_ret% = SqlExec(connectid%)
'Process the results of each statement.
 DO UNTIL i_ret% = NOMOREROWS
   i_ret% = SqlNextRow(connectid%)
   IF i_ret% = NOMOREROWS THEN Exit DO
   ELSE IF Result = REGROW THEN
      PRINT "a regular row was returned."
      PRINT
   ELSE
      'This row is a COMPUTE clause result.
      Length& = SqlADLen(connectid%, computeid%, 1)
      MyData$ = SqlAData(connectid%, computeid%, 1)
```

```
      PRINT MyData$+ " is " + Length& +  " bytes long."
END IF LOOP
```

Remarks

SqlADLen returns the string length of the data for a column in a COMPUTE clause. The format lengths returned for data of other datatypes are as follows:

Datatype	Length in bytes
bit	3
tinyint	3
smallint	6
timestamp	8
int	11
float	21
money	26
datetime	27
binary	255
varbinary	255
char	Length of the column (up to 255)
varchar	Length of the column (up to 255)
image	4096
text	4096

See Also

SqlAData, SqlAltLen, SqlNextRow, SqlNumAlts

SqlAltColId

Retuns the operand column ID for a compute column. The first column ID is 1.

Syntax

SqlAltColId (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets. This is obtained from the SqlNextRow function.

column%

The column number. The first column number returned is 1.

Results

When either computeid% or column% is invalid a -1 is returned. Normally it returns the id to which the aggregate operator in the compute clause pertains.

Example

```
computeid% = SqlNextRow(connectid%)
numops% = SqlNumOps(connectid%, computeid%)
For I = 1 to numops%
   colid% = SqlAltColId(connectid%, computeid%, I)
Next
```

Remarks

Calling SqlAltColId on the following would return 2 since the aggregate applies to the second column in the select clause.

```
select state, cust_name from customer
order by state, cust_name
compute count (cust_name) by state
```

SqlAltLen

Returns the maximum length of the data for a compute column in bytes. For the actual length of a column use the SqlADLen function.

Syntax

SqlAltLen (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

Is returned by the SqlNextRow function.

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets.

column%

The column number. The first column number returned is 1.

Results

The maximum integer number of bytes the compute clause result column can be. When the computeid% or the column% is invalid -1 is returned.

Example

```
computeid%=SqlNextRow(connectid)
numalts%=SqlNumAlts(connectid%, computeid%)
For I = 1 to numalts%
altlen%=SqlAltLen(connectid%, computeid%,I)
Next
```

Remarks

This applies particularly to variabe length data. Rather than the actual length, the maximum length is returned. SqlADLen should be used to obtain the actual length. The maximum length for each data type is as follows:

Datatype	Length in bytes
bit	3
tinyint	3
smallint	6
timestamp	8
int	11

float	20
money	26
datetime	27
binary	255
varbinary	255
char	Length of the column (up to 255)
varchar	Length of the column (up to 255)
image	4096
text	4096

SqlAltOp

Returns the type of aggregate function for a compute column.

Syntax

SqlAltOp (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

Is returned by SqlNextRow.

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets.

column%

The column number. The first column number returned is 1.

Results

The type of aggregate function operated on a particular column in a compute cluase. When computeid% or column% are invalid, -1 is returned.

Example

```
computeid%=SqlNextRow(connectid%)
numalts%=SqlNumAlts(connectid%, computeid%)
For I = 1 to numalts%
altop%=SqlAltOp(connectid%, computeid%,I)
Next
```

Remarks

These are the available aggregate functions types:

Aggregate function type	Aggregate operator
SQLAOPSUM	SUM
SQLAOPAVG	AVG
SQLAOPCNT	COUNT
SQLAOPMIN	MIN
SQLAOPMAX	MAX

To convert the integer type into a readable string use the function SqlPrType.

SqlAltType

Returns the datatype for compute column.

Syntax

SqlAltType (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

is returned by SqlNextRow.

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets.

column%

The column number. The first column number returned is 1.

Results

The value returned is the token value for a SQL Server datatype. If computeid% or column% is invalid, -1 is returned.

Example

```
computeid%=SqlNextRow(connectid%)
numalts%=SqlNumAlts(connectid%, computeid%)
For I = 1 to numalts%
   alttype%=SqlAltType(connectid%, computeid%,I)
Next
```

Remarks

```
select status, cust_name from customer
order by status, cust_name
compute count (cust_name) by status
```

SqlAltType would return SqlInit as the type of a count column. To convert the integer type to a meaningful string use the function SqlPrType.

SqlAltUType

Returns the user-defined datatype for a compute column.

Syntax

SqlAltUType (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets. This is returned by the SqlNextRow function.

column%

The column number. The first column number returned is 1.

Results

A token integer value for the user-defined datatype. -1 if either the computeid% or column% is invalid.

Example

```
computeid%=SqlNextRow (connectid%)
numalts%=SqlNumAlts(connectid%, computeid%)
For I = 1 to numalts%
   altutype%=SqlAltUType(connectid%, computeid%,I)
Next
```

Remarks

For a description of how to add user defined datatypes to the SQL Server database see either the Microsoft Transact-SQL Reference or Sybase Transact-Sql User's Guide.

SqlBCPBatchPC

Commits all rows sent via the SqlBCPSendRowPC to the destination table.

Syntax

bcp% = SqlBCPBatchPC ( )

Parameters

none

Results

SUCCEED (1) or FAIL (0)

Remarks

If this function is not used and an error occurs during processing, all the rows sent with the SqlBCPSendRowPC are not stored in the destination table.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPColumnsPC, SqlBCPDonePC, SqlBCPInitPC, SqlBCPSendRowPC, SqlBCPSetDataPC

SqlBCPColfmt

Specifies format information of a file in a bulk-copy operation.

Syntax

SqlBCPColfmt (connectid%, fcolumn%, ftype%, fplen%, fclen&, fterm$, ftlen%, tcol%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

fcolumn%

The column in the file for which the format is being specified. The first column is number 1.

ftype%

The datatype of the column in the file. If the datatype is different from the datatype of the parallel column in the database table (tcol%), the datatype is automatically converted.

To indicate the same datatype as in the corresponding column of the database table (tcol%) , equate this parameter to 0.

fplen%

The length prefix for the column in the file. Length prefixes may be 1, 2, and 4 bytes. To indicate no prefix, equate this parameter to 0. To indicate that the bcp utility should determine whether to use a length prefix, equate this parameter to -1. Bcp will use the necessary length prefix if the database column length is variable.

fclen&

The maximum length of this column's data in the file Do not include the length of any length prefix and/or terminator. Making fclen& equal to 0 indicates the data is NULL. Making fclen& equal to -1 instructs the system to ignore this parameter, IE no default maximum length.

For fixed-length datatypes, fclen& should be -1 except when the data is NULL, in which case fclen& should be 0.

For character, text, binary, and image data, fclen& can be -1, 0, or any value greater than 0.

fterm$

The terminator character sequence to be used for this column. To not use a terminator, equate this parameter to NULL (chr$(0)). To identify the tab character as the terminator, equate it to chr$(9).

ftlen%

The length, in bytes, of the terminator character sequence to be used for this column. When a terminator is not being used set this to -1.

tcol%

The corresponding column in the database table. If this value is 0, this column is not copied. The first column is column 1.

Returns

SUCCEED (1) or FAIL (0).

Remarks

Each call to SqlBCPColfmt describes the format for one column in the file.

SqlBCPColumns must be called before SqlBCPColfmt.

SqlBCPColfmt must be called for every column in the file.

See Also

SqlBCPColumnFormat,SqlBCPColumns, SqlBCPControl, SqlBCPExec, SqlBCPInit

SqlBCPColumnFormat

Specifies the column format information for the input file in a bulk-copy operation. This function combines SqlBCPColumns and SqlBCPColfmt in one step.

Syntax

SqlBCPColumnFormat(connectid%, col(), numcols%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

col( )

An array of a user-defined datatype (structure) defined in the .BI include file, as BCPColData, that contains the following elements:

ftype%

The datatype of the column in the file. If the datatype is different from the datatype of the parallel column in the database table (tcol%), the datatype is automatically converted.

To indicate the same datatype as in the corresponding column of the database table (tcol%), equate this parameter to 0.

fplen%

The length prefix for the column in the file. Length prefixes may be 1, 2, and 4 bytes. To indicate no prefix, equate this parameter to 0. To indicate that the bcp utility should determine whether to use a length prefix, equate this parameter to -1. Bcp will use the necessary length prefix if the database column length is variable.

fclen&

The maximum length of this column's data in the file. Do not include the length of any length prefix and/or terminator. Making fclen& equal to 0 indicates the data is NULL. Making fclen& equal to -1 instructs the system to ignore this parameter, IE no default maximum length.

For fixed-length datatypes, fclen& should be -1 except when the data is NULL, in which case fclen& should be 0.

For character, text, binary, and image data, fclen& can be -1, 0, or any value greater than 0.

fterm$

The terminator character sequence to be used for this column. To not use a terminator, equate this parameter to NULL (chr$(0)). To identify the tab character as the terminator, equate it to chr$(9).

ftlen%

The length, in bytes, of the terminator character sequence to be used for this column. When a terminator is not being used set this to -1.

tcol%

The corresponding column in the database table. If this value is 0, this column is not copied. The first column is column 1.

numcols%

The total number of columns to be copied.

Returns

SUCCEED (1) or FAIL (0).

Remarks

SqlBCPColumnFormat is the same as to calling SqlBCPColumns and repeated calls to SqlBCPColfmt.

The size of the col array is equal to the number of columns (numcols%). To access an element of col( ), you must use dot notation. eg:

Col.ftype%

See Also

SqlBCPColfmt, SqlBCPColumns

SqlBCPColumns

Sets the total number of columns in the operating-system file for a bulk-copy operation.

Syntax

SqlBCPColumns%(connectid%, colcount%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

colcount%

The total number of columns in the file.

Returns

SUCCEED (1) or FAIL (0).

Remarks

SqlBCPColumns can be called only after you call SqlBCPInit with a valid filename.

This function is for use only when you intend to use a format for an file that differs from the default format. For a description of the default format for an operating-system file, see SqlBCPInit.

See Also

SqlBCPColfmt, SqlBCPColumnFormat, SqlBCPInit

SqlBCPColumnsPC

Sets the number of columns to be sent to the server through the Bulk Copy facility. This function will bind numcol% variables to the columns in the destination database.

Syntax

ret% = SqlBCPColumnsPC(numcol%)

Parameters

numcol%

The number of columns in the destination table

Results

SUCCEED (1) or FAIL (0)

Example

```
ret% = SqlBCPColumnsPC ( 14 )
```

Remarks

If this number is less than the number of columns in the destination table then an error will occur. The indication of an error will be returned in the ret% field.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPDonePC, SqlBCPInitPC, SqlBCPSendRowPC, SqlBCPSetDataPC, SqlBCPBatchPC

SqlBCPControl

Changes the default settings for various control parameters for bulk-copy operations.

Syntax

SqlBCPControl(connectid%, param%, value&)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

param%

Constant	Description
BCPMAXERRS%	The number of errors allowed before terminating. The default is 10. Assigning a value of less than 1 to this field resets it to its default value.
BCPFIRST%	The first row to copy. The default is 1. Assigning a value of less than 1 to this field resets it to its default value.
BCPLAST%	The last row to copy. The default is to copy all rows. Assigning a value of less than 1 to this field resets it to its default value.
BCPBATCH%	The number of rows per batch. The default is 0. Assigning a value of less than 1 to this field resets it to its default value.

value&

The value for the specified param%.

Returns

SUCCEED (1) or FAIL (0).

Remarks

SqlBCPControl sets the control parameters for bulk-copies, such as the number of errors allowed before abandoning an operation.

See Also

SqlBCPColfmt,SqlBCPColumns, SqlBCPExec, SqlBCPInit

SqlBCPDonePC

Terminates the connection with the SQL server.

Syntax

ret% = SqlBCPDonePC( )

Parameters

none

Results

SUCCEED (1) or FAIL (0)

Remarks

All rows that have been sent to the server are now processed. The BCP process will be destroyed so do not attempt to use it again after this function has been used.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPColumnsPC, SqlBCPInitPC, SqlBCPSendRowPC, SqlBCPSetDataPC, SqlBCPBatchPC

SqlBCPExec

Executes a bulk-copy of data between a database table and an operating-system file.

Syntax

SqlBCPExec (connectid%, rowscopied&)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

rowscopied&

The number of rows successfully copied.

Returns

SUCCEED (1) or FAIL (0).

SqlBCPExec returns SUCCEED only when all rows are copied.

Remarks

SqlBCPExec copies data from an operating-system file to a database table or the other way around, depending on the value of the direction% parameter in SqlBCPInit.

You must call SqlBCPInit before calling SqlBCPExec.

See Also

SqlBCPColfmt, SqlBCPColumns, SqlBCPControl,SqlBCPInit

SqlBCPInit

Initializes a bulk-copy operation.

Syntax

SqlBCPInit(connectid%, tblname$, hfile$, errfile$, direction%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

tblname$

The name of the database table to copy.

hfile$

The name of the file to copy.

errfile$

The name of the error file to be used.

If NULL no error file is used.

direction%

The direction of the copy. This parameter must be one of two values: DBIN% or DBOUT%. DBIN% indicates a copy into the database table; DBOUT% indicates a copy from a database table.

Returns

SUCCEED (1) or FAIL (0).

Remarks

SqlBCPInit sets the default data formats for the file and examines the structure of the database table.

The default data formats are as follows:

- The order, type, length, and number of the columns in the file are identical to the order, type, length, and number of the columns in the database table.
- If the data in a given database column is of fixed length, then the data column in the file is also of fixed length.
- If the data in a given database column is of variable length or can contain NULL values, the data column in the file is prefixed by a 4-byte length value for SQLTEXT and SQLIMAGE datatypes and a 1-byte length value for all other types.
- There are no terminators between columns in the file.

See Also

SqlBCPColfmt, SqlBCPColumns, SqlBCPControl, SqlBCPExec, SqlBCPSetL

SqlBCPInitPC

Opens a Bulk Copy connection to the SQLserver.

Syntax

bcp% = SqlBCPInitPC(server$,userid$, password$, workstation$, application$, tablename$, errfile$)

Parameters

server$

The server identification

userid$

The userid for the connection.

password$

The password for the connection.

workstation$

The workstation id of the user (not required - can be "")

application$

The application name

tablename$

The destination tablename for the Bulk Copy.

errfile$

The DOS file name of the file to receive any errors which occur during the Bulk Copy.

Results

SUCCEED (1) or FAIL (0)

Example

```
bcp% = SqlBCPInitPC ("42NT","sa", "", "MyStation", "LoadPubs", "titles",
"C:\error.fil)
```

Remarks

The Userid and Password variables must be initialized prior to executing this command.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPColumnsPC, SqlBCPDonePC, SqlBCPSendRowPC, SqlBCPSetDataPC, SqlBCPBatchPC

SqlBCPReadFmt

Use this function to read a datafile format definition from a host file for use in Bulk Copy operations.

Syntax

ret% = SqlBCPReadFmt(connectid%, fname$)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

fname$

The filename of the file containing the format information.

Results

SUCCEED (1) or FAIL (0)

Remarks

After the function SqlBCPReadFmt is completed successfully DB-Library makes calls to bcp_columns and bcp_colfmt to automate the bulk copy of multiple files that share a common data format.

SqlBCPSendRowPC

Sends a row of data to the destination table through the Bulk Copy facility.

Syntax

ret% = SqlBCPSendRowPC( )

Parameters

none

Results

SUCCEED (1) or FAIL (0)

Remarks

The rows have not been committed. Stop sending on an error.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPColumnsPC, SqlBCPDonePC, SqlBCPInitPC, SqlBCPSetDataPC, SqlBCPBatchPC

SqlBCPSetDataPC

Sets the data values to be sent in string format.

Syntax

ret% = SqlBCPSetDataPC(col% ,datavalue$)

Parameters

col%

The column number.

datavalue$

The string representation of the data value to be sent.

Results

SUCCEED (1) or FAIL (0)

Remarks

This function is typically called for each row, each column. If a column value does not change it only has to be called once for that column. The datavalue is reset by a subsequent call.

All SqlBCP functions with a PC suffix are used for bulk-copying program data to SQL-Server. All the BCP functions without the PC suffix are used for bulk-copying files.

See Also

SqlBCPColumnsPC, SqlBCPDonePC, SqlBCPInitPC, SqlBCPSendRowPC, SqlBCPBatchPC

SqlBCPSetL

Sets the connection to allow or disallow bulk-copy operations.

Syntax

SqlBCPSetL(connectid%, enable%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

enable%

A Boolean value, TRUE (1) or FALSE (0), that specifies whether or not to enable bulk-copy operations for the SQL Server connection. By default, SQL Server connections are not enabled for bulk-copy operations.

Returns

SUCCEED (1) or FAIL (0).

Remarks

To keep users from initiating a bulk-copy sequence with SQL statements, avoid using this function in applications that permit ad hoc queries. Once a bulk-copy sequence begins, it cannot be stopped with an ordinary SQL statement.

See Also

SqlBCPInit

SqlBCPWriteFmt

Use this function to write a datafile format definition to a host file for use in Bulk Copy operations.

Syntax

ret% = SqlBCPWriteFmt(connectid%, fname$)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

fname$

The filename of the file to contain the format information.

Results

SUCCEED (1) or FAIL (0)

Remarks

The file created by this function can later be used in the function SqlBCPReadFmt.

SqlByList

Returns the binary string representing the column positions for the bylist in a compute clause.

Syntax

SqlByList (connectid%, computeid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

is returned by SqlNextRow.

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets.

column%

The column number. The first column number returned is 1.

Results

The resulting string contains one character per column position in the bylist.

Example

```
bylist$=SqlByList(connectid%, computeid%, column%)
```

Remarks

The number of bylist entries can be determined by getting the length of the result string. To get the column postition of the first column in the bylist ***Only call SqlByList when results exist.***

```
i% = ASC (MID$ (bylist$, 1, 1))
```

SqlCancel

Stops the execution of the statements which have been passed through the command buffer to the SQL Server.

Syntax

SqlCancel (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
ret_code% = SqlCancel(connectid%)
if ret_code% <> 1 then
   ...process errors
end if
```

Remarks

When you call SqlCancel, SQL Server terminates execution of all the statements associated with the connectid%. All results pending are read and discarded. SqlCancel would normally be called after statements like SqlExec, SqlOk, SqlSend, SqlResults, or SqlNextRow. In order to cancel only one statement out of several other statements in the command buffer, use SqlCanQuery.

See Also

SqlCanQuery, SqlExec, SqlNextRow, SqlOk, SqlResults, SqlSend

SqlCanQuery

Stops the execution of the currently executing statement which has been passed through the command buffer to the SQL Server.

Syntax

SqlCanQuery (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
ret_code% = SqlCanQuery(connectid%)
if ret_code% <> 1 then
   ... process errors
end if
```

Remarks

All pending results for the statement in execution are read and discarded. Using this function will achieve the same results as calling SqlNextRow until such a time as NOMOREROWS is returned.

See Also

SqlCancel, SqlNextRow, SqlResults, SqlSend

SqlChange

Returns the database name currently in use on the server if the database was changed during execution of a Transact-SQL batch.

Syntax

SqlChange (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

It returns the current database name as a string unless the database was not changed, where in that case an empty string will be returned.

Example

```
dbnamechange$=SqlChange(connectid%)
```

Remarks

SqlChange informs the application of a change from one database to another by detecting any occurence of the Transact-SQL USE statement.

Database changes made by a USE statement are in effect only at the end of the batch.

The simplest way to keep track of database switches is to call SqlChange when NOMORERESULTS is returned at the end of a command batch.

You can get the name of the current database by calling SqlName.

See Also

SqlExec, SqlName, SqlSend.

SqlClose

Closes the connection made with SqlOpenConnection.

Syntax

SqlClose (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

There is no value returned.

Example

```
SqlClose(connectid%)
```

Remarks

SqlClose terminates the activity of a SQL Server connection. It closes the connection and frees allocated memory. To close all open connections use SqlExit.

See Also

SqlExit, SqlOpenConnection

SqlClrBuf

Drops the specified number of rows from the row buffer.

Syntax

SqlClrBuf (connectid%, rows&)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

rows&

The number of rows to drop.

If rows& < 1 the call is ignored

If rows& > the number of rows in the buffer all but the newest is cleared.

Results

Nothing is returned from this function.

Example

```
SqlClrBuf(connectid%, 100)
```

Here, 100 signifies the number of rows you want cleared. All others are kept.

Remarks

In the case where the row buffer becomes full, SqlClrBuf can drop unwanted rows, beging with the oldest rows first. Rows are cleared on a first in/first out basis.

SqlClrOpt

Clears an option set by SqlSetOpt.

Syntax

SqlClrOpt (connectid%, option%, optparam$)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

option%

The option to be cleared.

The following is a list of options which may be set.

SQLARITHABORT

Aborts a query when an overflow or divide-by-zero error occurs during query execution. If SQLARITHABORT is not set, SQL Server substitutes null values and returns a warning message after the query has been executed. The default setting is off.

SQLARITHIGNORE

Substitutes null values when an overflow or divide-by-zero error occurs during a query. No warning message is returned. If SQLARITHIGNORE is not set, SQL Server substitutes null values and returns a warning message after the query has been executed. The default setting is off.

SQLBUFFER

Buffers result rows. SQLBUFFER is required when you use SqlGetRow. When you set SQLBUFFER, supply a parameter for the number of rows you want buffered. The default setting is 0 (no row buffering).

Parameter	Description
Less than 0	Buffer set to 100 rows.
0	No result rows buffered.
1	Not allowed.
2 - 32,767	The number of rows to buffer.

SQLNOAUTOFREE

Causes the command buffer to clear only with a call to SqlFreeBuf. When SQLNOAUTOFREE is not set, the first call to SqlCmd after a call to SqlExec or SqlSend, automatically clears the command buffer before new text is entered. The default setting is off.

SQLNOCOUNT

Stops returning information about the number of rows affected by each Transact-SQL statement. The default setting is off.

SQLNOEXEC

Processes a query through the compile step but does not execute it. You can use this option with SQLSHOWPLAN. Once SQLNOEXEC is set, no subsequent statements are executed until SQLNOEXEC is turned off. The default setting is off.

SQLOFFSET

Indicates whether SQL Server should return offsets to certain constructs in the query. This option takes a parameter that specifies the particular construct. Valid values of this parameter include:

select

from

table

order

compute

statement

procedure

execute

param (refers to parameter of a stored procedure)

Offsets are returned only if the batch contains no syntax errors.

SQLPARSEONLY

Checks the syntax of a query and returns appropriate error messages. The default setting is off.

SQLROWCOUNT

Specifies a maximum number of regular rows to be returned for SELECT statements. SQLROWCOUNT does not limit the number of compute rows returned.

When you set SQLROWCOUNT, supply a parameter for the number of rows you want returned. The default setting is 0, which returns all rows determined by SELECT statements.

Parameter	Description
0	Returns all rows generated by a SELECT statement.
1 - 2,147,483,647	The maximum # of regular rows to be returned for SELECT statements.

SQLSHOWPLAN

Generates a description of the processing plan. The default setting is off.

SQLSTAT

Returns performance statistics (CPU time, elapsed time, I/O) to the workstation after each query. The front end receives these statistics in the form of informational messages, and applications access them through a user-defined message handler. When you set SQLSTAT, supply a parameter for the type of performance statistics you want. The default setting is off.

Parameter	Description
IO	Returns statistics about SQL Server's internal I/O.
TIME	Returns information on parsing, compilation, and execution times (in millisecs)

SQLSTORPROCID

Sends the stored procedure ID to the workstation before sending rows generated by the stored procedure.

SQLTEXTLIMIT

When setting this option, supply a parameter that is the length, in bytes, of the longest text or image value your application can handle. To keep SQL Server from sending extra text, use the SQLTEXTSIZE option. The default setting is 4096.

Parameter	Description
0 - 32,768	Size, in bytes, of the longest text or image value your application can handle.

SQLTEXTSIZE

Limits the size of text or image values SQL Server returns. When setting this option, supply a parameter that is the length, in bytes, of the longest text or image value that SQL Server returns. The default setting is 4096.

Parameter	Description
0 - 32,768	Size, in bytes, of the longest text or image value SQL Server returns.

optparam$

A parameter for an option. Not all options have parameters. You must include optparam$ for all options, whether they take parameters or not. Any value for optparam$ can be supplied when using an option other than SQLSTAT. With the SQLSTAT option, you must include a valid parameter (either io or time).

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlClrOpt turns off options that have been set with SqlSetOpt. Options are cleared when you send the statements in the command buffer to SQL Server by invoking SqlExec. The results of the command generated by SqlClrOpt are not returned until the command is transferred to SQL Server. If an invalid parameter is specified, it is not detected until the command is sent to SQL Server and the results for that command are returned using SqlResults.

See Also

SqlSetOpt, SqlNextRow, SqlGetRow

SqlCmd

Adds Transact-SQL statements to the command buffer for processing.

Syntax

SqlCmd (connectid%, cmd$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

cmd$

A character string to be placed into the command buffer.

Results

SUCCEED (1) or FAIL (0)

Example

```
RetCode% = SqlCmd(connectid%, "SELECT name FROM sysobjects")
RetCode% = SqlCmd(connectid%, " WHERE id = 5")
RetCode% = SqlCmd(connectid%, " AND type = 'S'")
```

Remarks

SqlCmd% appends text to the existing command buffer. The appended text does not overwrite the current contents until the contents of the buffer are sent to SQL Server. After a call to SqlExec% or SqlSend%, the first call to SqlCmd% automatically clears the command buffer before the new text is entered. You can call SqlCmd% repeatedly. Sequential calls are appended, so ensure that you add the necessary blanks at the end of one line or the beginning of the next.

SqlCmdRow

Determines whether the current command can return rows.

Syntax

SqlCmdRow (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
ret_code%=SqlCmdRow(connectid%)
if ret_code% = 1 then
...process as if rows being returned
```

Remarks

If the current Transact-SQL statement in the command buffer is a SELECT statement or EXECUTE statement on a stored procedure containing a SELECT statement the variable ret_code will be set to 1 otherwise ret_code will be set to 0.

See Also

SqlNextRow, SqlResults, SqlRows

SqlColBrowse

Indicates whether the source of a result column can be updated using SQL Server browse-mode facilities.

Syntax

SqlColBrowse(connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The number of the result column. The first column is number 1.

Results

SUCCEED (1) or FAIL (0)

Remarks

You can call SqlColBrowse any time after you call SqlResults.

The SqlColBrowse function determines wheter the database column that is the source of the current result column can be updated using the SQL-Server browse-mode facilities. Only a column from a table that has a unique index and a timestamp column can be updated.

SqlColLen

Returns the maximum length in bytes of the data for a specific column.

Syntax

SqlColLen (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The value returned is the maximum number of bytes that the data for a particular column can hold. SqlColLen will return -1 if the column number is not within range.

Example

```
collen%=SqlColLen(connectid%)
```

Remarks

SqlColLen will return the maximum length that the data can be for that particular column, not the length of the data itself in the column. If you wish to know the actual data length in a column, SqlDatLen should be used.

The following table shows the maximum length of the data in a column for each datatype:

Datatype	**Length in bytes**
bit	3
tinyint	3
smallint	6
timestamp	8
int	11
float	21
money	26
datetime	27
binary	255
varbinary	255
char	Length of the column (up to 255)
varchar	Length of the column (up to 255)
image	4096
text	4096

See Also

SqlColName, SqlColType, SqlData, SqlDatLen.

SqlColName

Returns the column name for a specific column.

Syntax

SqlColName (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

A string containing the column name for a particular column. An empty string will be returned if the number of the column requested is out of the scope.

Example

```
colname$=SqlColName(connectid%,1)
The following uses SqlColName to return the name from the sysobjects
table:
'initialize the command buffer.
Cmd$ = "SELECT name, type FROM sysobjects"
'Send the statement to the SQL Server and begin execution.
SqlExec(connectid%)
'Process the statement results.
ret_code% = SqlResults(connectid%)
if ret_code% <> 1 then
   ... process errors
end if
'Print the column names.
For colnum% = 1 to 2
   column$ = SqlColName(connectid%, connum%)
   PRINT "Column";colnum%;" name returned  is "; Column$
Next colnum%
Output:
Column 1 name returned is name
Column 2 name returned is type
```

See Also

SqlColLen, SqlColType, SqlData, SqlDatLen.

SqlColSource

Returns the name of the database column to which the result column pertains.

Syntax

SqlColSource (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The value returned is the source name for a particular column or an empty string if the column if the column is the result of a Transact-SQL expression such as SUM(colname), or column% is out of range.

Example

```
colsource$=SqlColSource(connectid%, 1)
```

Remarks

You can call SqlColSource any time after a call to SqlResults

NOTE: SqlColSource returns the underlying column name not the optional column header you can specify with a select statement.

SqlColType

Returns the data type for a specific column.

Syntax

SqlColType (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The value returned is the column type for a particular column. If the number of the column is not in the specified range, -1 will be returned.

Example

```
coltype%=SqlColType(connectid%, 1)
```

Remarks

The integer value representing the column type is returned.

See Also

SqlColLen, SqlColName, SqlData.

SqlColUType

Returns the user defined data type for a specific column.

Syntax

SqlColUType (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The value returned is the user defined column type for a particular column. If the number of the column is not in the specified range, -1 will be returned.

Example

```
colutype%=SqlColUType(connectid%, 1)
```

Remarks

The integer value representing the user defined column type is returned.

See Also

SqlColLen, SqlColName, SqlData.

SqlCount

Returns the number of rows affected by the current statement.

Syntax

SqlCount(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of rows affected by the most recent statement.

Example

```
count%=SqlCount(connectid%)
```

Remarks

This function should only be called after the results of a transact-SQL statement are processed.

SqlCurCmd

Returns the number of the current statement in the command buffer.

Syntax

SqlCurCmd (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

Returns the number of the statement currently in the command buffer. The first statement is always 1.

Example

```
curcmd%=SqlCurCmd(connectid%)
```

Remarks

The first statement in a group of statements is 1. It increases every time a SUCCEED or FAIL is returned by SqlResults. A call to SqlExec or SqlSend will reset the count.

See Also

SqlCmdRow, SqlExec, SqlResults, SqlRows, SqlSend

SqlCurRow

Returns the number of the row most recently read.

Syntax

SqlCurRow (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of the current row.

Example

```
currow%=SqlCurRow(connectid%)
```

Remarks

The row number changes with every call to SqlNextRow. A call to SqlResults will reset the row number to 0.

SqlData

Retrieves a string representation of the column data for the last row retrieved.

Syntax

SqlData (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The return value is a string containing the data for a particular column.

When the column% is out of range on empty string is returned. When the data is NULL the string "NULL" is returned.

Example

```
dat$ = SqlData(connectid%, column%, 1)
```

Remarks

Use SqlDatLen to obtain the length of the data for variable length data types. Use SqlColType to obtain the datatype of a particular column.

SqlDataReady

Indicates if SQL Server is completed processing a command.

Syntax

SqlDataReady (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Remarks

SqlDataReady is a way to determine when statement processing is complete. Call SqlDataReady continuously until it returns a non-zero value, at which point you may call SqlOk. It should be noted that it is possible in conflicting lock situations, SQLDataReady will return FAIL every time.

See Also

SqlOk, SqlResults, SqlSend

SqlDateCrack

Converts a string of date and time values into a format more usable to the user.

Syntax

SqlDateCrack%(connectid%, dateinfovar, datetime$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

dateinfovar (a dateinfo variable)

Identifies a structure containing the components of the datetime$ string. The dateinfo structure is a user defined type found in the SQLDEF.BAS file. The structure contains the following fields:

Field	Description
year%	A number of a year in the range 1753 through 9999.
quarter%	A number of a quarter of a year in the range 1 though 4.
month%	A number of a month in the range 1 through 12.
dayofyear%	A number of a day of a year in the range 1 through 366. Leap years are counted.
day%	A number of a day of a month in the range 1 through 31.
week%	A number of a week of a year in the range 1 through 54. Leap years are counted.
weekday%	A number of the day of a week in the range 1 through 7 (Monday though Sunday).
hour%	A number of an hour in the range 0 through 23.
minute%	A number of a minute in the range 0 through 59.
second%	A number of a second in the range 0 through 59.
millisecond%	A number of a millisecond in the range 0 through 999.

datetime$

A string containing the date and time.

Results

SUCCEED (1) or FAIL (0).

Example

```
'Put the statement into the command buffer.
Result% = SqlCmd%(Sqlconn%, "SELECT name, crdate FROM
master..sysdatabases")
'Send the statement to SQL Server and start execution.
Result% = SqlExec%(Sqlconn%)
'Process the statement results.
Result% = SqlResults%(Sqlconn%)
```

```
'Retrieve and print the database name and its date info.
DO UNTIL SqlNextRow%(Sqlconn%) = NOMOREROWS
   PRINT "Database Name is "
   PRINT SqlData$(Sqlconn%, 1)
   PRINT
   PRINT "Creation date string info is "
   PRINT SqlData$(Sqlconn%, 2)
   PRINT
   'Break up the creation date into its constituent parts.
   Datetime$ = SqlData$(Sqlconn%, 2)
   SqlDateCrack(Sqlconn%, Dateinfo(), Datetime$)
   'Print the parts of the creation date.
   PRINT
   PRINT "Year = "; Dateinfo.year
   PRINT "Month = "; Dateinfo.month
   PRINT "Day of month = "; Dateinfo.day
   PRINT "Day of year = "; Dateinfo.dayofyear
   PRINT "Day of week = "; Dateinfo.weekday
   PRINT "Hour = "; Dateinfo.hour
   PRINT "Minute = "; Dateinfo.minute
   PRINT "Second = "; Dateinfo.second
   PRINT "Millisecond = "; Dateinfo.millisecond
LOOP
```

Remarks

SqlDateCrack converts a SQL Server DATETIME string into its integer components and puts them into a **dateinfo** structure.

Date and time values are maintained in an internal format that is not readily usable. For example, a time value is stored as the number of 300ths of a second since midnight, and a date value is stored as the number of days since January 1, 1900. The SqlDateCrack function is used to convert the internal value to something easily usable by an application.

See Also

SqlData

SqlDatLen

Returns the actual length in bytes of the data for a specific column.

Syntax

SqlDatLen (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The value returned is the actual column length for a particular column. If the value is NULL, 0 is returned. If the column does not exist -1 is returned.

Example

```
datalen%=SqlDatLen(connectid%, column%)
```

Remarks

SqlDatLen will return the maximum printable width for numeric datatypes such as small int and float. Use the Visual Basic Len function to determine the actual length.

The following table shows the length of the datatypes:

Datatype	**Length in bytes**
bit	3
tinyint	3
smallint	6
timestamp	8
int	11
float	21
money	26
datetime	27
binary	255
varbinary	255
char	Length of the column (up to 255)
varchar	Length of the column (up to 255)
image	4096
text	4096

To obtain the maximum length use SqlColLen. Use SqlData to return the data.

See Also

SqlColLen, SqlColName, SqlColType, SqlData.

SqlDead

Indicates if a SQL Server connection is inactive.

Syntax

SqlDead(connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlDead is useful in user supplied error handlers. If the SQL Server connection is dead, almost every procedure that receives that connection id as a parameter immediately fails.

SqlExec

Sends the SQL command(s) in the command buffer to the server and waits for the server to respond.

Syntax

SqlExec (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

FAIL will be returned if

- - the command buffer contains a syntax error
- - permission violation in the command
- - previous results pending
- - the command buffer is empty

Example

```
ret_code% = SqlExec(connectid%)
if ret_code% <>1 then
   ...process errors
end if
ret_code% = SqlResults(connectid%)
```

Remarks

SqlExec causes the Transact-SQL statements in the command buffer to be sent to SQL Server. Use SqlCmd to add statements to the buffer. Use SqlResults to process results from SqlExec.

See Also

SqlResults

SqlExit

Terminates all SQL Server connections.

Syntax

SqlExit

Parameters

none

Results

There is no value returned.

Example

```
SqlExit
```

See Also

SqlClose

SqlFirstRow

Returns the number of the first row in the row buffer.

Syntax

SqlFirstRow (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

The number of the first row in the row buffer. The first row returned is number 1.

Remarks

If row buffering is not turned on, SqlFirstRow, SqlCurRow, and SqlLastRow always return the same value: the number of the current row. If you turn on row buffering using SqlSetOpt with the SQLBUFFER option SqlFirstRow returns the number of the lowest row of results in the buffer. SqlLastRow returns the number of the result row stored in the highest (newest) buffer location.

See Also

SqlClrBuf, SqlCurRow, SqlGetRow, SqlLastRow, SqlNextRow, SqlSetOpt

SqlFreeBuf

Clears the command buffer.

Syntax

SqlFreeBuf (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

The command buffer is cleared.

Remarks

Statements are added to the command buffer with SqlCmd. After a call to SqlExec or SqlSend, the first call to SqlCmd automatically calls SqlFreeBuf to clear the command buffer before the new text is entered. If you don't want the buffer automatically cleared, set the SQLNOAUTOFREE option using SqlSetOpt. When SQLNOAUTOFREE is set, the command buffer is cleared only by a call to SqlFreeBuf.

See Also

SqlCmd, SqlExec, SqlSend, SqlStrCpy, SqlStrLen

SqlFreeLogin

Frees the memory allocated by SqlLogin for a login record.

Syntax

SqlFreeLogin(loginrec%)

Parameters

loginrec%

A login record. The value of loginrec% is returned by SqlLogin.

Remarks

You can call SqlFreeLogin immediately after you call SqlOpen or you can use the same login record for multiple calls to SqlOpen. Call SqlFreeLogin when you are completely finished with a login record.

SqlGetChar

Returns the value of a character in the command buffer.

Syntax

SqlGetChar (connectid%, charnum%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

charnum%

The character to find in the command buffer. The first character is at position 0. IE charnum% = 0.

Results

The character at position charnum% in the command buffer. If charnum% is not in range, an empty string is returned.

Remarks

Use SqlGetChar to get a particular character in the command buffer.

See Also

SqlCmd, SqlFreeBuf, SqlStrCpy, SqlStrLen

SqlGetOff

Used to check for the existence of Transact-SQL statements in the command buffer.

Syntax

SqlGetOff (connectid%, offset%, startpos%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

offset%

The type of offset you want to find. The choices are OFF_SELECT, OFF_FROM, OFF_ORDER, OFF_COMPUTE, OFF_TABLE, OFF_PROCEDURE, OFF_STATEMENT, OFF_PARAM, and OFF_EXEC.

startpos%

The point in the buffer from which begin searching. The command buffer begins at position 0.

Results

The beginning character position in the command buffer where a specified offset is found. If the offset is not found, -1 is returned.

Remarks

If the SQLOFFSET option has been set using SqlSetOpt, SqlGetOff can check for the location of certain Transact-SQL statements in the command buffer. SqlGetOff does not recognize SELECT statements in a subquery.

See Also

SqlCmd, SqlGetChar, SqlSetOpt, SqlStrCpy, SqlStrLen

SqlGetRow

Sets the current row in the row buffer to a specific row number and reads it.

Syntax

SqlGetRow(connectid%, row&)

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

row&

The number of the row to read. The first row returned is number 1.

Returns

One of four types of values depending on certain conditions.

- For a regular row, REGROW (-1) is returned.
- For a compute row, the ID of the COMPUTE clause is returned.
- If unsuccessful, FAIL (0) is returned..
- If the row does not exist, NOMOREROWS (-2) is returned..

Remarks

After calling SqlGetRow, make calls to SqlNextRow to return rows in order following the row accessed by SqlGetRow.

See Also

SqlClrBuf, SqlNextRow

SqlGetTime

Returns the number of seconds that the SQL-Sombrero/VBX will wait for SQL Server to respond to a Transact-SQL statement.

Syntax

SqlGetTime (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

The number of seconds that the front end waits for a SQL Server response to a Transact-SQL statement. 0 (the default) indicates an infinite time-out period.

Remarks

The time-out value can be changed by calling SqlSetTime.

See Also

SqlSetTime

SqlHasRetStat

Determines if a return status number was generated by the current Transact-SQL command or remote stored procedure.

Syntax

SqlHasRetStat (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0).

Remarks

Status numbers are a feature of stored procedures. Only a remote stored procedure or an EXECUTE statement can generate a status number. SqlRetStatus actually returns the status number. Stored procedures that complete normally return a status number equal to 0. When executing a stored procedure, the server returns the status number immediately after returning all other results. Therefore, the application can only call SqlHasRetStat after making the appropriate calls to SqlResults and SqlNextRow.

See Also

SqlNextRow, SqlResults, SqlRetData, SqlRetStatus

SqlInit

Initializes the SQL-Sombrero/VBX library.

Syntax

SqlInit( )

Parameters

none

Returns

A string containing the version number of the DB-Library being used. If unsuccessful, SqlInit returns an empty string.

Example

dblib$ = SqlInit ()

Remarks

SqlInit must be called before calling any other SQL-Sombrero/VBX functions.

See Also

SqlWinExit

SqlIsAvail

Determines whether a connection is available for use.

Syntax

SqlIsAvail (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

SUCCEED (1) if the connectid% is available for general use; otherwise, FAIL (0).

Remarks

SqlIsAvail indicates whether the specified connection is available for use.

SqlIsCount

Indicates whether or not the count returned by SqlCount is real.

Syntax

SqlIsCount (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
iscount=SqlIsCount(connectid%)
```

Remarks

CASE 1Some commands can affect rows and don't.

CASE 2Some commands cannot affect rows and still don't.

SqlCount would return 0 in both cases. SqlIsCount would return 1 for CASE 1 and 0 (FAIL) for CASE 2.

SqlIsOpt

Checks the status of an option set by SqlSetOpt.

Syntax

SqlIsOpt (connectid%, option%, optparam$)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

option%

The option to be checked.

optparam$

Sets the parameter of an option. See SqlClrOpt for more information on parameters.

Returns

SUCCEED (1) or FAIL (0).

Remarks

Although you can set and clear SQL Server query options directly through Transact-SQL, make sure your application uses SqlSetOpt and SqlClrOpt. With these functions, your application can make use of SqlIsOpt to determine the status of an option.

See Also

SqlClrOpt, SqlSetOpt

SqlLastRow

Returns the number of the last row in the row buffer.

Syntax

SqlLastRow (connectid%)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

Results

The number of the last row in the row buffer. The first row returned from SQL Server is number 1.

Remarks

If row buffering is not turned on, SqlFirstRow, SqlCurRow, and SqlLastRow always return the number of the current row. If you turn on row buffering with SqlSetOpt using the SQLBUFFER option, SqlLastRow returns the number of the highest (newest) row of results in the buffer.

See Also

SqlClrBuf, SqlCurRow, SqlFirstRow, SqlGetRow, SqlNextRow, SqlSetOpt

SqlLogin

Allocates a login record for use with SqlOpen.

Syntax

SqlLogin ( )

Parameters

none

Results

An integer identifier of a login record. If the login record cannot be allocated, 0 is returned.

Remarks

The following functions are used to supply the components of a login record:

- SqlSetLUser - the login ID (required).
- SqlSetLPwd - the user's password. (required only if the user has a password on Sql Server)
- SqlSetLHost - the workstation name (optional).
- SqlSetLApp - the application name (optional).

See Also

SqlOpen, SqlSetLApp, SqlSetLHost, SqlSetLPwd, SqlSetLUser

SqlMoreCmds

The SqlMoreCmds function indicates whether there are more statements in the command buffer that have yet to be processed.

Syntax

SqlMoreCmds%(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0). SUCCEED indicates there are more results in the command buffer to be processed.

Remarks

You can call the SqlMoreCmds function after the SqlNextRow function returns NOMOREROWS. You can also get the same information by calling the SqlResults function until it returns NOMORERESULTS.

See Also

SqlCmdRow, SqlResults, SqlRows, SqlRowType

SqlMoreText

Sends a portion of a large text or image value to SQL Server.

Syntax

SqlMoreText%(connectid%, datsize&, data$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

datsize&

The size, in bytes, of the text or image value being sent to SQL Server. You cannot send more text or image bytes to SQL Server than are specified in the call to SqlWriteText. The datsize& parameter cannot contain more than 32K characters of data.

data$

A string variable containing the text or image portion to be written to the SQL Server.

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlMoreText is used in conjunction with the SqlWriteText function to send a large text or image value to SQL Server in the form of a number of smaller chunks. You can only use SqlMoreText and SqlWriteText to perform updates to data on SQL Servers.

See Also

SqlName

Returns the database name currently in use on the server.

Syntax

SqlName (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The function returns current database name as a string.

Example

```
dbname$=SqlName(connectid%)
```

Remarks

SqlChange can keep you informed of all database changes.

See Also

SqlChange

SqlNextRow

Retrieves the next row of data from the server.

Syntax

SqlNextRow (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

- -1 or REGROW when a regular row is returned.
- 0 indicates a FAIL
- -2 or NOMOREROWS when there are no more rows to be read.

If the current row is a COMPUTE row the identification number of the COMPUTE clause is returned.

Example

```
ret_code% = SqlNextRow(connectid%)
```

Remarks

This function should be called to retrieve each row until the function returns -2 (NOMOREROWS).

Before making any calls to SqlNextRow, SqlResults must be called and it must return SUCCEED (1).

SqlNumAlts

Returns the number of columns in a compute row.

Syntax

SqlNumAlts (connectid%, computeid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

computeid%

The id representing the COMPUTE clause. A SELECT statement can have multiple COMPUTE clauses, which can have varying numbers of aggregate operators and aggregate targets. This is returned by the SqlNextRow function.

Results

The computeid is a value returned from the SqlNextRow function.

Example

```
computeid%=SqlNextRow(connectid%)
numalts%=SqlNumAlts(connectid%, computeid%)
```

Remarks

SqlResults should be called successfully prior to calling SqlNumAlts.

See Also

SqlResults, SqlNextRow

SqlNumCols

Returns the number of columns in the current set of results.

Syntax

SqlNumCols (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of columns in the current results set. 0 is returned if no columns exist.

Example

```
'Put 2 statements in the command buffer.
cmd$ = "SELECT name, type FROM sysobjects"
cmd$ = cmd$ + " SELECT name FROM sysobjects"
retcode% = SqlCmd(connectid%, cmd$)
'Send the statements and execute them.
retcode% = SqlExec(connectid%)
'Process the results of both statements.
DO UNTIL SqlResults(connectid%) = NOMORERESULTS
   PRINT SqlNumCols(connectid%);
   PRINT "columns."
   DO UNTIL SqlNextRow(connectid%)=NOMOREROWS
   'Code to process the data your way
   LOOP
LOOP
Outputis :2 columns.1 columns.
```

Remarks

SqlNumCols refers to the number of output columns.

See Also

SqlColLen, SqlColName

SqlNumCompute

Returns the number of COMPUTE clauses in the current set of results.

Syntax

SqlNumCompute% (connectid%)

Parameter

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of COMPUTE clauses present in the current results set.

Example

```
numcompute%=SqlNumCompute(connectid%)
If the select statement was
   select status, cust_name from customer
   order by status, cust_name
   compute count(cust_name) by status
   compute count(cust_name)
numcompute% above would equal 2.
```

Remarks

Ensure that the SqlResults function is called before attempting the SqlNumCompute function.

See Also

SqlNumAlts, SqlResults

SqlNumOrders

Returns the number columns specified in the ORDER BY clause in the current set of results.

Syntax

SqlNumOrder (connectid%)

Parameter

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of columns is the ORDER BY clause.

Example

```
numorder%=SqlNumOrder(connectid%)
```

Remarks

SqlResults must be called before SqlNumOrder and it must return SUCCEED.

See Also

SqlOrderCol, SqlResults

SqlNumRets

Returns the number of returned parameter values generated by a stored procedure.

Syntax

SqlNumRets(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of parameters returned by the most recently executed stored procedure.

Example

```
numrets%=SqlNumRets(connectid%)
```

Remarks

Sql Server returns stored procedure parameter values immediately after all other results. This means if the stored procedure executes a select, the results of the select are returned before the parameter value results. Therefore, SqlResults and SqlNextRow must be called as many times as necessary prior to the call to SqlNumRets.

See Also

SqlNextRow, SqlResults, SqlRetData, SqlRetLen, SqlRetName, SqlRetType

SqlOk

Verifies if the SQL command(s) in the command buffer are correct.

Syntax

SqlOk (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

A FAIL is most commonly received due to syntax errors.

Example

```
ret_code% = SqlOk(connectid%)
if ret_code% <> 1 then
   ...process errors
end if
```

Remarks

The combination of SqlSend and SqlOk is the equivalent of the SqlExec function. After a SUCCEED status from SqlSend, SqlOk must be called. When a SUCCEED is returned, the results can be processed by calling SqlResults.

See Also

SqlSend, SqlExec, SqlResults, SqlNextRow

SqlOpen

Allocate and initialize a SQL Server connection.

Syntax

SqlOpen(loginrec%, server$)

Parameters

loginrec%

A login record. The value of loginrec% is returned by SqlLogin.

server$

The name of the SQL Server you want to connect to.

Results

The identifier of A SQL Server connection. SqlOpen returns 0 if a SQL Server connection cannot be created or initialized, or if your login to SQL Server fails. When 0 is returned, the Error Handler is called to indicate the error.

Remarks

SqlOpen allocates a data structure for the connection, initiates communication with the network, logs in to SQL Server, and sets any default options. SqlOpen returns an id that is used by almost every function. A program can open multiple connections with SQL Server. The same login record can be used for multiple calls to SqlOpen.

SqlOpen returns 0 when it encounters any of the errors in the following:

Error	Description
SQLECONN	Server is unavailable or does not exist.
SQLEPWD	Login is incorrect.
SQLESQLPS	Maximum number of connections already allocated.

See Also

SqlClose, SqlExit, SqlLogin, SqlSetLoginTime

SqlOpenConnection

Opens the connection to the SQL server.

Syntax

SqlOpenConnection(server$, loginid$, pwd$, workstation$, app$)

Parameters

server$

The name of the SQL Server you want to connect to.

loginid$

The login id.

pwd$

The password.

workstation$

The workstation name up to 30 characters. (may be " ")

app$

The application name up to 30 characters.

Results

Returns a Connection object if the connection to the server was made. 0 is returned if no connection can be made.

Example

```
connectid% = SqlOpenConnection (server$, loginid$, pwd$, workstation$,
app$)
```

Remarks

SqlOpenConnection performs the actions required to establish a connection to the server.

You should not modify the identifier returned by SqlOpenConnection. Modifying the identifier can cause unexpected results.

SqlOpenConnection allocates a data structure for the connection, initiates communication with the network, logs in to SQL Server, and sets any default options. SqlOpen returns an id that is used by almost every function. A program can open multiple connections with SQL Server. The same login record can be used for multiple calls to SqlOpen.

SqlOpen returns 0 when it encounters any of the errors in the following:

Error	Description
SQLECONN	Server is unavailable or does not exist.

SQLEPWD	Login is incorrect.
SQLESQLPS	Maximum number of connections already allocated.

SqlOrderCol

Returns the ID of a column appearing in the most recently executed query's ORDER BY clause.

Syntax

SqlOrderCol (connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The column number. The first column number returned is 1.

Results

The column number (based on position in select list) for a column in the ORDER BY clause.

Example

```
num% = SqlNumOrder(connectid%)
For I=1 to num%
ordercol%=SqlOrderCol(connectid%, I)
...
Next
```

Remarks

Select cust_name, cust_id, status from customer

order by status, cust_name

A call to SqlOrderCol with a parameter of 1 will return 3 because status is first in the order by clause and 3rd in the select list.

SqlPrType

Returns the printable data type for a specific column type.

Syntax

SqlPrType (connectid%, coltype%)

Parameter

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

coltype%

The parameter passed is the column type to get the printable type for.

Results

The value returned is the column type in a printable (string) format for a particular column type.

Example

coltype% = SqlColType(connectid%, column%)

prtype$ = SqlPrType(connectid%, coltype%)

Remarks

SqlColType returns a Sql Server token value for a column type. SqlPrType returns a more humanly readable form of the token value.

Token value	Datatype
SQLINT1	tinyint
SQLINT2	smallint
SQLINT4	int
SQLMONEY	money
SQLMONEY4	smallmoney
SLQFLT4	real
SLQFLT8	float
SQLDATETIME	datetime
SQLDATETIM4	smalldatetime
SQLBIT	bit
SQLCHAR	char
SQLVARCHAR	varchar
SQLTEXT	text
SQLBINARY	binary
SQLVARBINARY	varbinary
SQLIMAGE	image

SQLINTN	integer-null
SQLDATETIMN	datetime-null
SQLMONEYN	money-null
SQLFLTN	float-null
SQLAOPSUM	sum
SQLAOPAVG	avg
SQLAOPCNT	count
SQLAOPMIN	min
SQLAOPMAX	max

See Also

SqlAltType, SqlColType

SqlQual

Returns a string representing the WHERE clause for the current row in a specified table.

Syntax

SqlQual (connectid%, tabnum%, tabname$)

Parameters

connectid%

The parameter passed is the connection id returned from the SqlOpenConnection function.

tabnum%

Specifies an integer representing the number of the table. Tables are numbered in the order they are listed in the FROM clause. Table numbers start at 1. If tabnum% is -1, tabname$ is used to identify the table.

tabname$

A string containing the name of a table specified in the FROM clause. If tabname$ is an empty string, tabnum% is used to identify the table.

Results

A string containing the WHERE clause for the current row in a specified table. If the specified table cannot be browsed, SqlQual returns an empty string. A browseable table has a unique index and a timestamp column.

Remarks

SqlQual provides a WHERE clause that can be used to update a single row in a browseable table. Columns from this row must have been previously retrieved through a browse-mode SELECT query. The WHERE clause produced by SqlQual begins with the keyword WHERE and contains references to the row's unique index and timestamp column. You can simply append the WHERE clause to an UPDATE or DELETE statement; there is no need to examine it or manipulate it in any way.

The timestamp column indicates the time that a particular row was last updated. An update on a browseable table fails if the timestamp column in the WHERE clause that SqlQual generates is different from the timestamp column in the table. Such a condition, which generates SQL Server error message 532, indicates that another user updated the row since it was selected for browsing. Design your application to include the logic for handling an update failure.

SqlQual can construct WHERE clauses only for browseable tables. You can use SqlTabBrowse to determine whether a table can be browse. SqlQual is usually called after SqlNextRow.

See Also

SqlColSource

SqlResults

Tells the Sql Server to execute the next staement in the SQL command buffer sent by the SqlExec function.

Syntax

SqlResults (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1), FAIL (0) or NOMORERESULTS (2).

Example

```
ret_code% = SqlResults(connectid%)
```

Remarks

This function should be called as many times as there are statements to be executed in the command buffer. (The example in this package is set up for only one SQL statement). It is called after a SUCCEED status of SqlExec or SqlOk. It will always return a status of SUCCEED or NOMORERESULTS the first time it is called if a SUCCEED was returned from SQLExec or SqlOk. To process result rows after SqlResult returns a SUCCEED status, SqlNextRow would be used.

Regardless if the statement returns rows or not, you must call SqlResults for every statement found in the command buffer.

See Also

SqlExec, SqlOk, SqlRows, SqlNextRow

SqlRetData

Retrieves a string representation of a value returned from a Stored Procedure.

Syntax

SqlRetData(connectid%, retnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

retnum%

The number corresponding to a return parameter of a stored procedure in the order specified by the create procedure. Non return parameters are not counted. If a stored procedure has two parameters and only the second is a return parameter, retnum% would be 1. (For remote stored procedures, the order is not necessarily the same as the Create Procedure.)

Results

A string containing the data for a specified retnum%. An empty string will be returned if the parameter is not in the scope specified. The string "NULL" will be returned for NULL results.

Example

```
dat$=SqlRetData(connectid%,2)
```

Remarks

The number of returns available can be determined by using the function SqlNumRets. To function as a return parameter, a parameter must be declared as OUTPUT in both the Create Procedure statement and the EXECUTE statement. If a stored procedure is invoked with an EXECUTE statement, the return parameter values are only available if the command batch containing the EXECUTE statement uses variables rather than constants, for the return parameters.

See Also

SqlNextRow, SqlNumRets, SqlResults, SqlRetLen, SqlRetName, SqlRetType

SqlRetLen

Returns the length in bytes of a return parameter value generated by the execution of a stored procedure.

Syntax

SqlRetLen (connectid%, retnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

retnum%

The number corresponding to a return parameter of a stored procedure in the order specified by the create procedure. Non return parameters are not counted. If a stored procedure has two parameters and only the second is a return parameter, retnum% would be 1. (For remote stored procedures, the order is not necessarily the same as the Create Procedure.)

Results

The length of the specified retnum%. If retnum% is out of range, -1 is returned. If the value is NULL, 0 is returned.

Example

```
numrets% = SqlNumRets(connectid%)
For I = 1 to numrets%
   retlen%=SqlRetLen(connectid%,I)
   ...
Next I
```

Remarks

The number of returns available can be determined by using the function SqlNumRets.

To function as a return parameter, a parameter must be declared as OUTPUT in both the Create Procedure statement and the EXECUTE statement.

If a stored procedure is invoked with an EXECUTE statement, the return parameter values are only available if the command batch containing the EXECUTE statement uses variables rather than constants, for the return parameters.

See Also

SqlNextRow, SqlNumRets, SqlResults, SqlRetData, SqlRetName, SqlRetType

SqlRetName

Returns the name of a stored procedure return parameter .

Syntax

SqlRetName (connectid%, retnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

retnum%

The number corresponding to a return parameter of a stored procedure in the order specified by the create procedure. Non return parameters are not counted. If a stored procedure has two parameters and only the second is a return parameter, retnum% would be 1. (For remote stored procedures, the order is not necessarily the same as the Create Procedure.)

Results

The name of the parameter specified by retnum%. An empty string is returned if retnum% is out of range.

Example

```
numrets% = SqlNumRets(connectid%)
For I = 1 to numrets%
   retname$=SqlRetName(connectid%,I)
   ...
Next I
```

Remarks

The number of returns available can be determined by using the function SqlNumRets.

To function as a return parameter, a parameter must be declared as OUTPUT in both the Create Procedure statement and the EXECUTE statement.

If a stored procedure is invoked with an EXECUTE statement, the return parameter values are only available if the command batch containing the EXECUTE statement uses variables rather than constants, for the return parameters.

See Also

SqlNextRow, SqlNumRets, SqlResults, SqlRetData, SqlRetLen, SqlRetType

SqlRetStatus

Returns a status number for the current stored procedure or remote stored procedure.

Syntax

SqlRetStatus(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

A normal completion of a stored procedure returns 0.

Example

```
retstatus& = SqlRetStatus(connectid%)
```

Remarks

The following table lists the other possible values and their meanings:

Value	Meaning
-1	Missing object.
-2	Datatype error.
-3	Process was chosen as deadlock victim.
-4	Permission error.
-5	Syntax error.
-6	Miscellaneous user error.
-7	Resource error, such as out of space.
-8	Nonfatal internal problem.
-9	System limit was reached.
-10	Fatal internal inconsistency.
-11	Fatal internal inconsistency.
-12	Table or index corrupt.
-13	Database corrupt.
-14	Hardware error.

Status numbers are a feature of stored procedures, therefore only an EXECUTE statement can generate a status number.

The server returns a status number immediately after returning all other results. A stored procedure can generate several results sets ie. one for each SELECT statement it contains, therefore, an application must call SqlResults% and SqlNextRow% as many times as required to process all the results before it can call any functions that process returned parameters.

The order in which the application processes the status number and any returned parameter values is

unimportant.

See Also

SqlNextRow, SqlNumRets, SqlResults, SqlRetData, SqlRetLen, SqlRetType, SqlRetName

SqlRetType

Returns the datatype of a return parameter generated by a stored procedure execution.

Syntax

SqlRetType (connectid%, retnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

retnum%

The number corresponding to a return parameter of a stored procedure in the order specified by the create procedure. Non return parameters are not counted. If a stored procedure has two parameters and only the second is a return parameter, retnum% would be 1. (For remote stored procedures, the order is not necessarily the same as the Create Procedure.)

Results

The returned value is the token value for the datatype.

The token value may not correspond directly to the column's SQL Server datatype. See the following:

Column	Returns
SQLVARCHAR	SQLCHAR
SQLVARBINARY	SQLBINARY
SQLDATETIMN	SQLDATETIME or SQLDATETIM4
SQLMONEYN	SQLMONEY or SQLMONEY4
SQLFLTN	SQLFLT8 or SQLFLT4
SQLINTN	SQLINT1, SQLINT2, or SQLINT4

Example

```
numrets% = SqlNumRets(connectid%)
For I = 1 to numrets%
   rettype%=SqlRetType(connectid%,I)
   ...
Next I
```

Remarks

The number of returns available can be determined by using the function SqlNumRets.

To function as a return parameter, a parameter must be declared as OUTPUT in both the Create Procedure statement and the EXECUTE statement.

If a stored procedure is invoked with an EXECUTE statement, the return parameter values are only available if the command batch containing the EXECUTE statement uses variables rather than constants,

for the return parameters.

See Also

SqlNextRow, SqlNumRets, SqlResults, SqlRetData, SqlRetLen, SqlRetName

SqlRows

This function indicates whether or not the last statement executed returned rows.

Syntax

SqlRows (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
ret_code% = SqlRows(connectid%)
if ret_code% = 1 then
...
```

Remarks

This should be executed after the call to SqlResults and prior to calling SqlNextRow.

See Also

SqlCmdRow, SqlNextRow, SqlResults

SqlRowType

This function indicates the type of row which has been returned. The row type returned is either a result row or a compute row.

Syntax

SqlRowType(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

- - REGROW(-1) if a regular row has been returned.
- - for a COMPUTE row the ID of the COMPUTE clause
- - FAIL(0) if function call was unsuccessful
- - NOMOREROWS(-2) if no rows have be read

Remarks

Since the function **SqlNextRow** returns the row type there is usually no reason to call this function.

See Also

SqlNextRow

SqlRpcInit

This function is used to initialize a remote stored procedure.

Syntax

SqlRpcInit(connectid%,rprocname$,optmask%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

rprocname$

This parameter is a string containing the name of the stored procedure to be initialized.

optmask%

This is a type byte variable containing a bitmask of options to be used in the initialization of the stored procedure. There is at present only one option available. The option name is SQLRPCRECOMPILE.

Results

SUCCEED (1) or FAIL (0)

Remarks

SQLRPCRECOMPILE - causes the stored procedure to be recompiled prior to execution.

There are two methods of calling a stored procedure from within an application. The first method is to execute a command buffer which contains an EXECUTE Transact-SQL statement.

There is a second method which involves using the SQL-Sombrero/VBX functions: SqlRpcInit, SqlRpcParam and SqlRpcSend. To call a remote stored procedure:

- Use the SqlRpcInit function to indicate which stored procedure to be invoked.

- Use the SqlRpcParam function to supply the parameters required to execute the stored procedure if necessary.

- Use the SqlRpcSend to first indicate that no further parameters are to be sent and also to inform the SQL Server to begin the execution of the stored procedure.

Results are retrieved from the SQL Server using the same functions as regular result rows. Use of the functions SqlRetData and SqlRetStatus is required at the end of the processing of the stored procedure's results.

Note that commands executed using remote stored procedures cannot be rolled back.

See Also

SqlNextRow, SqlResults, SqlRetData, SqlRetStatus, SqlRpcParam, SqlRpcSend, SqlOk

SqlRpcParam

This function is used to pass parameters to remote stored procedures.

Syntax

SqlRpcParam(connectid%,paramname$,optmask%,datatype
%,maxparamlenght&,actualparamlenght&,paramvalue$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

paramname$

This parameter is a string containing the name of the parameter that is being invoked for the stored procedure named in the last SqlRpcInit function.

optmask%

This parameter is a one byte variable containing parameter options for the stored procedure. At present the only option available is SQLRPCRETURN.

datatype%

This parameter contains the type of the parameter. The values are the same as returned by **SqlColType.**

maxparamlength&

This parameter contains the maximum length of data that can be passed back for a parameter passed back from a stored procedure. This parameter is only required for datatypes whose length can change such as text or image.

This parameter should be set to (-1) in the following cases:

- for fixed length datatypes
- if restricting the length would make no difference
- if the parameter is not a return parameter

actualparamlength&

This parameter contains the actual length of a paramter sent to the stored procedure. This does no include a null terminator.

This parameter should be set to (-1) in the following cases:

- for fixed length datatypes
- for return parameters

This parameter should be set to 0 if the parameter is an empty string.

paramvalue$

This parameter contains the actual data being sent as a parameter. If the actualparamlength& is 0 this

field is ignored. SQL-Sombrero/VBX will convert the string to its native datatype.

Results

SUCCEED (1) or FAIL (0)

Remarks

SQLRPCRETURN - this parameter is a return value from the stored procedure.

See SqlRpcInit for a discussion on the sequence of events to execute a remote stored procedure.

See Also

SqlNextRow, SqlResults, SqlRetData, SqlRetStatus, SqlRpcInit, SqlRpcSend, SqlOk

SqlRpcSend

This function indicates that there are no more parameters being sent and that the SQL Server should start executing the stored procedure.

Syntax

SqlRpcSend(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Remarks

See SqlRpcInit for a discussion on the sequence of events to execute a remote stored procedure.

See Also

SqlNextRow, SqlResults, SqlRetData, SqlRetStatus, SqlRpcInit, SqlRpcParam, SqlOk

SqlRPwClr

This function clears all remote passwords from the specified login rec.

Syntax

SqlRPwClr(loginrec%)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

Results

None

Remarks

The function **SqlRPwSet** will set a password needed to execute a stored procedure on another SQL Server. The user may specify more than one password using the **SqlRPwSet** function. There would be one password for each SQL Server which is called. This function will clear all passwords.

See Also

SqlLogin, SqlOpen, SqlRPwSet, SqlSetLApp, SqlSetLHost, SqlSetLPwd, SqlSetLUser

SqlRPwSet

This function adds a remote password to the login rec.

Syntax

SqlRPwSet(loginrec%,servername$,password$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

servername$

This parameter contains the name of the SQL Server which this password pertains to.

password$

This parameter contains the password used to connect to the SQL Server with the name passed in the servername$ field.

Results

SUCCEED (1) or FAIL (0)

Remarks

SQL Servers can execute stored procedures resident on other SQL Servers. To perform this action the first SQL Server logs on to the SQL Server on which the stored procedure is resident.

To allow the connection the SqlRPwSet function allows the user to specify the password to be used when opening a connection on the second SQL Server. Applications can specity a unique password for every remote SQL Server that will be logged on to.

See Also

SqlLogin, SqlOpen, SqlRPwClr, SqlSetLApp, SqlSetLHost, SqlSetLPwd, SqlSetLUser

SqlSend

Sends the SQL command(s) in the command buffer to the server and does not wait for the server to respond.

Syntax

SqlSend (connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

SUCCEED (1) or FAIL (0)

Example

```
ret_code% = SqlSend(connectid%)
if ret_code% <>1 then
...process errors
end if
```

Remarks

SqlSend will send Transact-SQL statements to the SQL Server that are stored in the command buffer. If you wish to add statements to the command buffer, you may also use SqlCmd. Once a SUCCEED status is returned, SqlOk is needed to confirm the correctness of statements found in the command buffer. Once that has been achieved, SqlResults must be called in to manipulate the results.

See Also

SqlOk, SqlResults, SqlExec, SqlNextRow

SqlSendCmd

Sends Transact-SQL text from the command buffer and sets up the statement for processing. This utility combines several functions into a single call.

Syntax

SqlSendCmd(connectid%, command$)

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

commandd$

A character string to be copied into the command buffer.

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlSendCmd performs several actions that process the Transact-SQL statements in the command buffer. SqlSendCmd is equivalent to calling SqlCmd, SqlExec, and SqlResults. SqlSendCmd prepares the command for processing the first set of results but does not retrieve the first data row.

Example

```
'Put commands into the command buffer.
cmd$ = "SELECT db_name(dbid), dbid, size FROM sysusages"
cmd$ = cmd$ + " ORDER BY dbid"
'Send commands to SQL Server and start execution.
Result% = SqlSendCmd%(SqlConn%,Cmd$)
```

See Also

SqlCmd, SqlExec, SqlResults

SqlServerEnum

Lists the names of local SQL Servers, network SQL Servers, or both.

Syntax

SqlServerEnum(searchmode%, servernames$, numsrventries%)

Parameters

searchmode%

An integer value which is used to determine whether SqlServerEnum checks for server names locally, on the network, or both. The constants which can be used for this parameter are LOCSEARCH and NETSEARCH.

CONSTANT	DESCRIPTION	Val
LOCSEARCH	searches for the server names listed in the WIN.INI file.	1
NETSEARCH	searches for server names on the network type that is defined by the default Net-Library..	2
LOCSEARCH + NETSEARCH	searches for server names using both methods,	3

servernames$

The buffer that stores the server names that SqlServerEnum returns. You must preallocate this buffer using an appropriate Visual Basic function such as Space$. Each server name is followed by a CHR(0).

numsrventries%

An output parameter that will contain the number of server names that were copied to the buffer by the SqlServerEnum function.

Results

One or more of the following status codes:

Status	Value
ENUMSUCCESS%	0
MOREDATA%	1
NETNOTAVAIL%	2
OUTOFMEMORY%	3
NOTSUPPORTED%	4

Remarks

The SqlServerEnum function returns a list of server names to the servername$ buffer in the order in which it finds them. Only complete server names are copied to the buffer. Each server name is separated by a null character (chr$(0)).

See Also

SqlOpen and SqlOpenConnection

SqlSetAvail

Marks a SQL Server connection as being available for general use.

Syntax

SqlSetAvail(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

None.

Remarks

Any subsequent call to SqlIsAvail returns SUCCEED until some use is made of the connectid% connection. SqlSetAvail is not normally required in an application.

See Also

SqlIsAvail

SqlSetLApp

Sets the application name in the SQL Server login record.

Syntax

SqlSetLApp(loginrec%,application$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

application$

This a string of up to 30 characters to be sent to the SQL Server indicating the application name associated with a connection.

Returns

SUCCEED(1) or FALSE(0)

Remarks

This function must be called prior to calling the SqlOpen function using the login rec id. If this function is not used the application name is an empty string.

This string is used to identify your application.

See Also

SqlLogin, SqlOpen, SqlSetLApp, SqlSetLPwd, SqlSetLUser

SqlSetLHost

Sets the workstation name in the SQL Server login record.

Syntax

SqlSetLHost(loginrec%,host$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

host$

This a string of up to 30 characters to be sent to the SQL Server indicating a host name for the connection.

Returns

SUCCEED(1) or FALSE(0)

Remarks

This function must be called prior to calling the SqlOpen function using the login rec id. If this function is not used the workstation name is an empty string.

This string is used to identify your workstation.

See Also

SqlLogin, SqlOpen, SqlSetLHost, SqlSetLPwd, SqlSetLUser

SqlSetLNatLang

Sets the national language in the SQL Server login record.

Syntax

SqlSetLNatLang(loginrec%,langname$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

langname$

This a string of up to 30 characters to be sent to the SQL Server indicating the name of the national language to use for the connection.

Returns

SUCCEED(1) or FALSE(0)

Remarks

This function must be called prior to calling the SqlOpen function using the login rec id. If this function is not used the language used is the SQL Server default language.

This string is used to set your choice of a language other than the SQL Server default language. Setting this parameter will cause error messages to be passed back in the language chosen.

See Also

SqlLogin, SqlOpen

SqlSetLoginTime

Sets the amount of time that you wish to wait while the VBX issues a SqlOpen. The time is given in seconds.

Syntax

SqlSetLoginTime(seconds%)

Parameters

seconds%

The parameter passed is the number of seconds to wait before the server will pass back a timeout notification.

Returns

SUCCEED(1) or FALSE(0)

Remarks

To indicate to the server that you will wait an infinite amount of time the **seconds%** parameter should be set to 0. Sixty (60) seconds is the default value for this parameter.

See Also

SqlSetTime

SqlSetLPwd

Sets the users password in the SQL Server login record.

Syntax

SqlSetLPwd(loginrec%,password$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

password$

This a string of up to 30 characters to be sent to the SQL Server indicating the password to be used for the connection.

Returns

SUCCEED(1) or FALSE(0)

Remarks

This function must be called prior to calling the SqlOpen function using the login rec id. If this function is not used the password used is an empty string.

This string is used to set the password for the userid. The passwords are kept in the syslogins table in the master database. The system administrator maintains the passwords for the SQL Server.

See Also

SqlLogin, SqlOpen, SqlSetLHost, SqlSetLApp, SqlSetLUser

SqlSetLUser

Sets the user id in the SQL Server login record.

Syntax

SqlSetLUser(loginrec%,userid$)

Parameters

loginrec%

The parameter passed is the loginrec id returned from the SqlLogin function.

userid$

This a string of up to 30 characters to be sent to the SQL Server indicating the user id to be used for the connection.

Returns

SUCCEED(1) or FALSE(0)

Remarks

This function must be called prior to calling the SqlOpen function using the login rec id.

This string is used to set the user id for a connection. You must provide a user id in order to establish a connection to the SQL Server. You must also use the SqlSetLPwd to provide the password asscociated with this user id.

See Also

SqlLogin, SqlOpen, SqlSetLHost, SqlSetLPwd, SqlSetLApp

SqlSetMaxProcs

Sets the maximum number of connections to the SQL Server that can be open at one time.

Syntax

SqlSetMaxProcs(maxnumprocs%)

Parameters

maxnumprocs%

The parameter passed is the new limit to the number of open connections to the SQL Server.

Returns

SUCCEED(1) or FALSE(0)

Remarks

The maximum open connections that the Visual Basic application can support is 45. The default number before using this function is 25 open connections.

See Also

SqlOpen

SqlSetOpt

Sets a SQL Server option.

Syntax

SqlSetOpt(connectid%,option%,parameter$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

option%

This is the option to be set. The list of possible values appears below.

parameter$

This is the parameter for the option being set. Some of the options being set can use a parameter. If the option being set does not require a parameter this field can be ignored and can be either an empty string "" or any other string value.

Returns

SUCCEED(1) or FALSE(0)

Remarks

If the parameter required is a string representation of a numerical value this parameter cannot have a leading space. This leading space will occur if the **Str$** Visual Basic function is used to obtain this string. Use the Visual Basic **Ltrim** function to remove the leading space.

Some of the options can be set or cleared using Transact-SQL. It is recommended to use the Visual Basic method of setting and clearing options. Three of the following options are unique to the Visual Basic application: **SQLBUFFER, SQLTEXTLIMIT** and **SQLNOAUTOFREE.**

If the option being set is equivalent to a Transact-SQL SET statement SQL-Sombrero/VBX places the corresponding SET statement into the command buffer. The commands in the buffer must be executed prior to the option taking effect. If the parameter being sent is no valid for the chosen option it will not be determined until the statement is sent to the SQL Server for execution.

The following is the list of options that can be set using this function.

SQLARITHABORT

This option if set will abort a query if a divide by zero error occurs or if an overflow ocurs.

If this option is not set NULL values are substitued and a warning message is sent after execution of the query.

There are no parameters for this option.

The default for this option is OFF.

SQLARITHIGNORE

This option if set will supress the issuing of warning messages when divide by zero or overflows occur.

There are no parameters for this option.

The default for this option is OFF.

SQLBUFFER

This option will set the number of rows which are allowed for buffering This is an option which can only be set using this function as there is no Transact_SQL equivalent.

The following parameters are valid for this option.

Parameter	**Description**
Less than 0	Buffer is set to 100 rows
0	No rows are buffered
1	Invalid value
2-32,767	Number of buffered rows

SQLNOAUTOFREE

This option is used to tell SQL-Sombrero/VBX that the command buffer will be cleared only by using the function **SqlFreeBuf.**

There are no parameters for this option.

The default for this option is OFF.

SQLNOCOUNT

This option will stop SQL Server from reporting the number of rows affected by each Transact-SQL statement.

There are no parameters for this option.

The default for this option is OFF.

SQLNOEXEC

Setting this option will cause the query to be compiled but not executed This option is in effect until being reset using the **SqlClrOpt** function.

There are no parameters for this option.

The default for this option is OFF.

SQLOFFSET

Set this option to indicate to SQL Server where to return offsets to certain constructs in the query.

The following is the list of valid values for the parameter to this option:

OFF_SELECT OFF_FROM

OFF_TABLE OFF_ORDER

OFF_COMPUTE OFF_STATEMENT

OFF_PROCEDURE OFF_EXEC

OFF_PARAM

Offsets are only returned if no syntax errors are present in the batch.

SQLPARSEONLY

Set this option to perform a syntax check only and return any error messages to the workstation.

There are no parameters for this option.

The default for this option is OFF.

SQLROWCOUNT

Setting this option will restrict the number of rows returned for all SELECT statements

If you use 0 then all rows will be returned. A number from 1 - 2,147,483,647 will restrict the number of rows returned to that value.

The default setting is 0 to allow all rows to be returned.

You can also set the parameter to 0 by using the **SqlClrOpt** function.

SQLSHOWPLAN

Set this option to have the SQL Server return the processing plan prior to executing the query.

There are no parameters for this option.

The default for this option is OFF.

SQLSTAT

Set this option to have the SQL Server return inormation about the execution of the query. This information is returned as SQL Server messages.

There are two parameter to be used with this option:

IO Returns statistics about the SQL Server internal I/O. Also returned are the number of table scans, the number of logical/physical reads and the number of pages written for each statement.

TIME Returns the amount of time taken to perform the parsing, compilation, and the actual execution time. The time returned indicates the number of milliseconds taken.

The default for this option is OFF.

SQLSTORPROCID

Setting this will cause the SQL Server to send the stored procedure ID to the workstation prior to sending rows back to the workstation.

There are no parameters for this option.

The default for this option is OFF.

SQLTEXTLIMIT

Set this option to limit the amount of text or image data which is sent to the application. SQL-Sombrero/VBX will read all text or image data but will ignore any part of that data which exceeds the limit set by this option.

The parameter sent is the largest number of bytes that the application can handle. Valid values are from 0 to 32,768.

The default value is 4096.

SQLTEXTSIZE

Set this option to limit the amount of text or image data which is sent to the workstation. SQL Server will send only the amount of data which is specified in the parameter.

The parameter sent is the largest number of bytes that the application can handle. Valid values are from 0 to 32,768.

The default value is 4096.

See Also

SqlClrOpt, SqlIsOpt

SqlSetTime

Sets the amount of time that you wish to wait while the VBX issues a SqlExec or SqlOk. The time is given in seconds.

Syntax

SqlSetTime(seconds%)

Parameters

seconds%

The parameter passed is the number of seconds to wait before the server will pass back a timeout notification.

Returns

SUCCEED(1) or FALSE(0)

Remarks

To indicate to the server that you will wait an infinite amount of time the **seconds%** parameter should be set to 0. 0 seconds (infinity) is the default value for this parameter.

See Also

SqlExec, SqlGetTime, SqlOk, SqlSend, SqlSetLoginTime

SqlStrCpy

Copies data from the command buffer into a program variable.

Syntax

SqlStrCpy (connectid%, start%, numbytes%, mybuff$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

start%

The starting byte position to begin copying from

numbytes%

The number of bytes to copy.

mybuff$

A string variable in which to store the result.

Results

SUCCEED (1) or FAIL (0) FAIL is returned if start% is < 0

Example

```
ret_code% = SqlStrCopy(connectid%,1,100,mybuff$)
if ret_code% <>1 then
...process errors
end if
```

This example copys the first 100 bytes from the command buffer into mybuff$.

Remarks

Internally the command buffer is a linked list of text strings.

See Also

SqlStrLen

SqlStrLen

Returns the number of bytes of data available in the command buffer.

Syntax

SqlStrLen(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of bytes in the command buffer.

The return value can be used in the SqlStrCpy function.

Example

strlen%=SqlStrLen(connectid%)

Remarks

Internally the command buffer is a linked list of text strings.

See Also

SqlStrCpy

SqlTabBrowse

Indicates whether a specified table can be updated with the SQL-Sombrero/VBX browse-mode procedures.

Syntax

SqlTabBrowse(connectid%, tablenumber%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

tablenumber%

Indicates the number of the table as specified in the SELECT statement's FROM clause. Table numbers start at 1.

Results

SUCCEED (1) or FAIL (0). If you drop a table's unique index while browsing, SqlTabBrowse continues to return SUCCEED.

Remarks

SqlTabBrowse is a SQL Server browse-mode function. SqlTabBrowse provides a way to identify browseable tables. A browseable table requires both a unique index and a timestamp column. SqlColBrowse is useful when examining ad hoc queries prior to performing browse-mode updates based on them. When a query is hard-coded into the application, the SqlTabBrowse function is unnecessary.

You can call SqlTabBrowse any time after you call SqlResults.

See Also

SqlColBrowse, SqlColSource, SqlQual, SqlTabCount, SqlTabName, SqlTabSource

SqlTabCount

Returns the number of tables included in the current SELECT statement.

Syntax

SqlTabCount%(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The number of tables, including SQL Server work tables, included in the current SELECT statement. If an invalid connectid% value is sent to SqlTabCount%, a value of --1 will be returned.

Remarks

SqlTabCount is a SQL Server browse-mode function. A SELECT statement can generate a set of result rows whose columns are derived from several database tables. To perform browse-mode updates of the columns in a statement's select list, your application must know how many tables are involved in the query, because each table requires a separate UPDATE statement. SqlTabCount can provide this information for ad hoc queries. When a query is hard-coded into the application, SqlTabCount is unnecessary.

You can call SqlTabCount any time after you call SqlResults.

See Also

SqlColBrowse, SqlColSource, SqlQual, SqlTabCount, SqlTabName, SqlTabSource

SqlTabName

Returns the name of a table based on its number.

Syntax

SqlTabName$(connectid%, tabnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

tabnum%

The number of a table. Table numbers start with 1.

Returns

A string containing the name of a specified table. This string is empty if the table number is out of range or if the specified table is a SQL-Server work table.

SqlTabSource

Returns the name and number of the table from which a result column

derives.

Syntax

SqlTabSource$(connectid%, column%, tabnum%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The number of the result column. Column numbers start at 1.

tabnum%

An integer variable to receive the table number. Many Visual Basic for SQL-Server functions that operate in browse mode accept either a table name or a table number. If SqlTabSource$ returns an empty string, tabnum% is set to -1.

Results

A string containing the name of the table from which a result column

derives. If an empty string is returned, it means one of the following:

- The SQL-Server connection is inactive. This is an error that invokes an application's error handler.
- The SELECT statement does not contain the FOR BROWSE clause.
- The column number is not in range.

SqlTsNewLen

Returns the length of the new value of a timestamp column after a browse-mode update.

Syntax

SqlTsNewLen(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The length, in bytes, of the updated row's new timestamp value. SqlTsNewLen returns --1 if no timestamp is returned to the application because the update was unsuccessful or because the UPDATE statement did not contain a WHERE clause returned by SqlQual.

Remarks

SqlTsNewLen is a Visual Basic for SQL Server browse-mode function. SqlTsNewLen provides information about the timestamp column. The WHERE clause returned by SqlQual contains references to the row's unique index and timestamp column. When you use such a WHERE clause in an UPDATE statement, a new value is placed in the updated row's timestamp column and a new timestamp value is returned to the application (if the update is successful). With SqlTsNewLen, the application saves the length of the new timestamp value, possibly for use with SqlTsPut.

See Also

SqlColBrowse, SqlColSource, SqlQual, SqlTabBrowse, SqlTabCount, SqlTabName, SqlTabSource, SqlTsNewVal, SqlTsPut

SqlTsNewVal

Return the identifier of the new value of a timestamp column after a browse-mode update.

Syntax

SqlTsNewVal(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

The identifier of the updated row's new timestamp value. The identifier is an empty string if no timestamp is returned to the application because the update was unsuccessful or because the UPDATE statement did not contain a WHERE clause returned by SqlQual. Important: Do not modify the identifier in any way. Modifying the identifier can cause unpredictable results.

Remarks

SqlTsNewVal and SqlTsNewVal are Visual Basic for SQL Server browse-mode functions. SqlTsNewVal provides information about the timestamp column. When used in an UPDATE statement, the WHERE clause returned by SqlQual places a new value in the updated row's timestamp column and returns the new timestamp value to the application (if the update is successful). With SqlTsNewVal, the application saves the new timestamp value, possibly for use with SqlTsPut.

See Also

SqlColBrowse, SqlColSource, SqlQual, SqlTabBrowse, SqlTabCount, SqlTabName, SqlTabSource, SqlTsNewLen, SqlTsPut

SqlTsPut

Puts the new value of the timestamp column into a specified table's current row in the row buffer.

Syntax

SqlTsPut(connectid%, newts$, newtslen%, tabnum%, tabname$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

newts$

The new timestamp value. The new timestamp value is returned by SqlTsNewVal.

newtslen%

The length of the new timestamp value. The length of the new timestamp value is returned by SqlTsNewLen.

tabnum%

The number of the table to receive the new timestamp. Table numbers start at 1. The tabnum% parameter must refer to a browseable table. Use SqlTabBrowse to determine whether the table you specify can be browsed. If the table is browseable, tabname$ is used to identify the table.

tabname$

A string containing the table name. The tabname$ parameter must refer to a browseable table. If the string is empty, tabnum$ is used to identify the table. The value of tabname$ is returned by SqlTabSource.

Results

SUCCEED (1) or FAIL (0). The following conditions cause SqlTsPut to return FAIL:

- The application tries to update the timestamp of a nonexistent row.
- The application tries to update the timestamp using an empty string as the new timestamp identifier (newts$ or newts&).
- The specified table cannot be browsed.

Remarks

SqlTsPut is a Visual Basic for SQL Server browse-mode function. SqlTsPut% manipulates the timestamp column. When used in an UPDATE statement, the WHERE clause returned by SqlQual places a new value in the updated row's timestamp column and returns the new timestamp value to the application (if the update is successful). If the same row is updated a second time, the UPDATE statement's WHERE clause must use the latest timestamp value. SqlTsPut updates the timestamp in the row currently being browsed. Then, if the application has to update the row a second time, it calls SqlQual to formulate a

new WHERE clause that uses the new timestamp. With SqlTsNewVal, the application saves a new timestamp value, possibly for use with SqlTsPut.

See Also

SqlColBrowse, SqlColSource, SqlQual, SqlTabBrowse, SqlTabCount, SqlTabName, SqlTabSource, SqlTsNewLen, SqlTsNewVal

SqlTsUpdate

Updates the value of a timestamp column in a specified table.

Syntax

SqlTsUpdate(connectid%, tabnum%, tabname$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

tabnum%

The number of the table to receive the new timestamp. Table numbers start at 1. The tabnum% parameter must refer to a browseable table. Use SqlTabBrowse to determine whether the table can be browsed. If this value is 1, the tabname$ parameter is used to identify the table.

tabname$

A string containing the table name. The tabname$ parameter must refer to a browseable table. If the string is empty, tabnum% is used to identify the table. The value of tabname$ is returned by SqlTabSource$.

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlTsUpdate is equivalent to calling SqlTsNewVal and SqlTsPut. Like those functions, it is designed for use in browse mode. SqlTsUpdate can update the timestamp column if either tabnum% or tabname$ is provided. If neither is provided, SqlTsUpdate causes an error.The connectid% connection must be clear--that is, it cannot have any rows pending. After completing all the updates to the timestamp column, immediately close the connectid% connection.

See Also

SqlTabBrowse, SqlTabSource, SqlTsNewVal, SqlTsPut

SqlTxPtr

Return the identifier for a text or image column in the current row.

Syntax

SqlTxPtr(connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The number of the column. The first column in a table is number 1.

Results

The identifier for a text or image column in the current row. In the case of a null text or image value, the identifier value is an empty string. Important: Do not modify this identifier in any way. Modifying the identifier can cause unpredictable results.

Remarks

Every text or image column has an associated identifier that uniquely identifies the text or image value. This identifier is useful in conjunction with SqlWriteText. The identifier returned by SqlTxPtr supplies the value for the textptr$ parameter of SqlWriteText.

See Also

SqlTxTimeStamp, SqlWriteText

SqlTxTimeStamp

Return the identifier for the text timestamp for a column in the current row.

Syntax

SqlTxTimeStamp(connectid%, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

column%

The number of the column. The first column in a table is number 1.

Results

A string containing the identifier of the text timestamp for the column. This identifier can be an empty string.Important: Do not modify this identifier in any way. Modifying the identifier can cause unpredictable results.

Remarks

Every text or image column has an associated text timestamp that marks the time of the column's last modification. The text timestamp is useful in conjunction with SqlWriteText% to ensure that two competing users do not inadvertently wipe out each other's modifications in the database.

See Also

SqlTxPtr, SqlWriteText

SqlTxTsNewVal

Return the identifier for the new value for a text timestamp after a call to SqlWriteText.

Syntax

SqlTxTsNewVal(connectid%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

Results

A string containing the identifier for the timestamp value for the text or image value modified by a SqlWriteText operation. This identifier can be an empty string. Important: Do not modify this identifier in any way. Modifying the identifier can cause unpredictable results.

Remarks

Every text or image column has an associated text timestamp that is updated whenever the column's value is changed. The new text timestamp identifier, returned by SqlTxTsNewVal can be used in conjunction with SqlWriteText to ensure that two competing users do not inadvertently wipe out each other's modifications in the database. After each successful SqlWriteText operation (which can include a number of calls to SqlMoreText), SQL Server sends the updated value of timestamp$ back to SQL-Sombrero/VBX. The application can then get the new value of timestamp$ with SqlTxTsNewVal and then use SqlTxTsPut to put that new value into the row buffer for future access through SqlTxTimeStamp. This capability is particularly useful when the application does not need the new timestamp immediately because row buffering is turned on.

See Also

SqlMoreText, SqlTxTimeStamp, SqlTxTsPut, SqlWriteText

SqlTxTsPut

Places the identifier for the new value for a text timestamp into a column of the current row in the row buffer.

Syntax

SqlTxTsPut(connectid%, newtxts$, column%)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

newtxts$

The new text timestamp value returned by SqlTxTsNewVal.

column%

The number of the column to receive the new text timestamp. Column numbers start at 1.

Results

SUCCEED (1) or FAIL (0).

Remarks

Every text or image column has an associated text timestamp that is updated whenever the column's value is changed. The text timestamp is useful in conjunction with SqlWriteText to ensure that two competing users do not inadvertently wipe out each other's modifications in the database. After each successful SqlWriteText operation (which can include a number of calls to SqlMoreText), SQL Server sends the updated text timestamp value back to SQL-Sombrero/VBX. SqlTxTsNewVal enables the application to get this new timestamp value. The application can then use SqlTxTsPut to place the new timestamp value in the row buffer for future access through SqlTxTimeStamp. This is particularly useful when the application does not need the new timestamp immediately because row buffering is turned on.

See Also

SqlMoreText, SqlTxTimeStamp, SqlTxTsNewVal, SqlWriteText

SqlUse

Sets the current database for a particular SQL Server connection.

Syntax

SqlUse(connectid%, dbname$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

dbname$

A string containing the database name.

Returns

SUCCEED (1) or FAIL (0).

Example

```
ret%=SqlUse(connectid%, dbname$)
```

Remarks

SqlUse emulates the Transact-SQL USE statement. If the USE statement fails because the requested database was still in a recovery process, SqlUse continues to send USE commands at 1-second intervals until it succeeds or until it encounters another type of error. If you call SqlCmd to place statements into the command buffer, execute these statements before calling SqlUse.

See Also

SqlExec, SqlResults

SqlWinExit

This functon is required to free memory allocated for SQL-Sombrero/VBX.

Syntax

SqlWinExit

Parameters

none

Results

There is no value returned.

Example

```
SqlWinExit
```

Remarks

This should be part of all exit routines.

SqlWriteText

Sends a text or image value to SQL Server.

Syntax

SqlWriteText(connectid%, objname$, textptr$, textptrlen%, timestamp$, log%, size&, text$)

Parameters

connectid%

The parameter passed is the connection id passed from the SqlOpenConnection function.

objname$

The database table and column name. The table name and the column name are separated by a period.

textptr$

The text or image value to be modified. This identifier can be obtained by calling SqlTxPtr.

textptrlen%

This parameter is the length of the text pointer and is included for future compatibility. At present its value must be the constant SQLTXPLEN which has a value of 16.

timestamp$

The text timestamp for the text or image value to be modified. This identifier can be obtained by calling SqlTxTimestamp. The value changes whenever the text or image value itself is changed.

log%

A Boolean value that specifies whether this SqlWriteText operation should be recorded in the transaction log. Valid values are TRUE or FALSE.

size&

The total size, in bytes, of the text or image value to be written.

text$

A string containing the text or image to be written. If this string is empty, the SQL-Sombrero/VBX expects the application to call SqlMoreText one or more times until all **size&** bytes of data have been sent to SQL Server. No single data block can be larger than 64K.

Results

SUCCEED (1) or FAIL (0).

Remarks

SqlWriteText is used to update text and image values, allowing an application to send long values to SQL Server without having to copy them into a Transact-SQL UPDATE statement. In addition, SqlWriteText gives an application access to the text timestamp mechanism, which can be used to ensure

that two competing users do not inadvertently wipe out each other's modifications in the database.

SqlWriteText succeeds only if its **timestamp$** parameter matches the text column's timestamp in the database. If a match occurs, SqlWriteText updates the text column and at the same time updates the column's timestamp. This has the effect of governing updates by competing applications--an application's SqlWriteText call will fail if a second application has updated the text column between the time the first application retrieved the column and the time it made its SqlWriteText call. SqlWriteText is similar to a Transact-SQL WRITETEXT statement. However, calling SqlWriteText is usually more efficient than sending a WRITETEXT statement through the command buffer. (For information about WRITETEXT, see the Microsoft SQL Server Transact-SQL Reference.)SqlWriteText can be invoked with or without logging in, according to the value of the log% parameter. To use SqlWriteText with logging turned off, the SQL Server option select into / bulkcopy must be set to TRUE by executing the following system procedure:

sp_dboption 'mbsql', 'select into/bulkcopy', 'true'

SqlWriteText, used in conjunction with SqlMoreText, also enables an application to send a large text or image value to SQL Server in the form of a number of smaller chunks. This is particularly useful with operating systems that are unable to allocate extremely long data buffers.When SqlWriteText is used with SqlMoreText, it locks the specified database text column, and the lock is not released until the final SqlMoreText has sent its data. This ensures that a second application does not read or update the text column in the middle of the first application's update.

If the text$ string is not an empty string, SqlWriteText executes the data transfer from start to finish, including any necessary calls to SqlOk and SqlResults. To send a text or image value in chunks rather than sending the whole value at once, set the text$ parameter to an empty string. SqlWriteText returns control to the application immediately after notifying SQL Server that a text transfer is about to begin. The actual text is sent to SQL Server with SqlMoreText, which can be called multiple times, once for each chunk.

See Also

SqlMoreText, SqlTxPtr, SqlTxTimeStamp, SqlTxTsNewVal, SqlTxTsPut