Visual FoxPro ODBC Driver Overview

Visual FoxPro is a powerful object-oriented environment for database construction and application development. The Microsoft Visual FoxPro ODBC Driver enables applications to open, query and update data in Visual FoxPro and earlier versions of FoxPro through the Open Database Connectivity (ODBC) interface.

For example, with the Microsoft Visual FoxPro ODBC Driver you can:

- Use Microsoft Query to query and update Visual FoxPro data from Microsoft Excel worksheets.
- Create mail-merge letters using Visual FoxPro data with Microsoft Word.
- Query and update Visual FoxPro views and tables from Microsoft Access.
- Use Visual FoxPro as the data store for Visual Basic, Visual C++ and C applications.

You can use the driver to accomplish many other tasks. The following table lists a few topics to help you get started.

To	See
Find out more about using Visual FoxPro data with Microsoft Office	Accessing Visual FoxPro data from a Microsoft Office Application
Learn about using Visual FoxPro data in Visual Basic applications	Using the Visual FoxPro ODBC Driver with your Visual Basic Application
View a simple example using Visual C++ to access Visual FoxPro data	Using the Visual FoxPro ODBC Driver with your C or C++ Application
See a list of supported hardware and software	System Requirements
Read an overview of the system	Overview of Driver Architecture

Setting Up the Visual FoxPro ODBC Driver

You use the Visual FoxPro ODBC Driver Setup program to:

- Add new components.
- Remove installed components.
- Reinstall to restore missing files and settings.
- Remove all previously installed components.

To set up the Visual FoxPro ODBC Driver

1. Run Setup.exe from Disk 1 of the Visual FoxPro ODBC Driver.
2. Follow the instructions on the screen.

Once you install the driver on your system, the Setup program recognizes the installed driver components and presents additional dialog boxes that enable you to change your driver's configuration.

Accessing Visual FoxPro Data from a Microsoft Office Application

You can use the Microsoft Visual FoxPro ODBC Driver to access Visual FoxPro data from your Microsoft Office for Windows 95 or Windows 98 applications.

To	See
Use Microsoft Access	Querying and Updating Visual FoxPro Data from Access
Use Microsoft Excel	Importing Data into Microsoft Excel from a Visual FoxPro Database
Use Microsoft Word	Creating Mailing Labels in Microsoft Word Using Visual FoxPro Data

Adding a Visual FoxPro Data Source

To access Visual FoxPro data from your application, you must have a data source. You can create a data source from

- Within an application, such as Microsoft Word, Microsoft Excel, or Microsoft Access, that uses ODBC drivers.
- Outside of your application, using the Windows 95, Windows 98, or Windows NT Control Panel.

Once a data source exists on your system, you can reuse the same data source each time you want to access Visual FoxPro data. If you have several different databases or tables you want to access, you can create a separate data source for each database or directory.

The following procedure creates a data source using the Control Panel. For more information on creating a data source from an application, see Accessing Visual FoxPro Data from a Microsoft Office Application.

To add a Visual FoxPro data source

1. Choose the **32bit ODBC** icon from the Windows 95, Windows 98, or Windows NT Control Panel. This option is available after you've installed the Visual FoxPro ODBC Driver, or any ODBC driver software.
2. In the **Data Sources** dialog box, click **Add**.
3. In the **Add Data Source** dialog box, select Microsoft Visual FoxPro Driver from the **Installed ODBC Drivers** list, then click **OK**.
4. In the **ODBC Visual FoxPro Setup** dialog box, enter the data source name, select the database type, select the database or directory, and click **OK**.

   The new data source name is displayed in the **User Data Sources (Driver)** list in the **Data Sources** dialog box.
5. In the **Data Sources** dialog box, click **Close**.

Modifying a Visual FoxPro Data Source

You can modify a FoxPro data source.

To modify a Visual FoxPro data source

1. Choose the **32bit ODBC** icon from the Windows 95, Windows 98, or Windows NT Control Panel. This option is available after you've installed the Visual FoxPro ODBC Driver, or any ODBC driver software.
2. In the **Data Sources** dialog box, select the name of the data source you want to modify in the **Data Sources (Driver)** list and click **Setup**.
3. In the **ODBC Visual FoxPro Setup** dialog box, select and change the items you want to modify, and then click **OK**.
4. In the **Data Sources** dialog box, click **Close**.

The changes you made are saved and take effect the next time you access the data source from your application.

Deleting a Visual FoxPro Data Source

You can delete a FoxPro data source.

To delete a Visual FoxPro data source

1. Choose the **32bit ODBC** icon from the Windows 95, Windows 98, or Windows NT Control Panel. This option is available after you've installed the Visual FoxPro ODBC Driver, or any ODBC driver software.
2. In the **Data Sources** dialog box, select the name of the data source you want to delete in the **Data Sources (Driver)** list.
3. Click **Delete**.

   The data source is no longer listed in the **Data Sources** dialog box.
4. Click **Close**.

Connecting to a Visual FoxPro Data Source

You can connect to a Visual FoxPro data source using your Microsoft Office application or using the SQL API.

To connect from	See
Microsoft Access, Microsoft Excel, or Word	Accessing Visual FoxPro Data from a Microsoft Office Application.
Your C or C++ application	SQLConnect SQLDriverConnect
Your Visual Basic application	Using the Visual FoxPro ODBC Driver with your Visual Basic Application

Using Connection Strings

You can use a connection string to connect to a Visual FoxPro data source.

For example, to connect to the TasTrade data source and override the current setting of Exclusive associated with the data source, you would use the string:

```
DSN=TasTrade;Exclusive=Yes
```

For a list of the attribute keywords and values you can include in the connection string, see **SQLDriverConnect**.

For a complete explanation of connection string syntax, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Accessing a Visual FoxPro Data Source from Microsoft Excel

If you have Microsoft Query installed, you can create a data source in Microsoft Excel that connects to Visual FoxPro data.

To access Visual FoxPro data from Microsoft Excel

1. Open a Microsoft Excel spreadsheet.
2. From the **Data** menu, choose **Get External Data**. Microsoft Query opens.
3. In the **Select Data Source** dialog box, click **Other**.
4. In the **ODBC Data Sources** dialog box, click **New**.
5. In the **Add Data Source** dialog box, select **Microsoft Visual FoxPro Driver** from the **Installed ODBC Drivers** list box and click **OK**.
6. In the **ODBC Visual FoxPro Setup** dialog box, enter the data source name, select the Database type, enter the path to the database or directory, and click **OK**.

   The new data source name is displayed in the **Enter Data Source text box** of the **ODBC Data Sources** dialog box.
7. Click **OK**.

   The new data source name is selected in the **Available Data Sources** text box of the **Select Data Source** dialog box.
8. Click **Use**.

You can now add tables to the open query. For more information on building a query, see Importing Data into Microsoft Excel from a Visual FoxPro Database.

Importing Data into Microsoft Excel from a Visual FoxPro Database

You can import Visual FoxPro data into your Microsoft Excel worksheet if you have defined a data source for it. For information about creating a Visual FoxPro data source, see Accessing a Visual FoxPro Data Source from Microsoft Excel.

To import Visual FoxPro data into an Microsoft Excel worksheet

1. Open a Microsoft Excel spreadsheet.
2. From the **Data** menu, choose **Get External Data**.

   Microsoft Query opens.
3. In the **Select Data Source** dialog box, select a Visual FoxPro data source, then click **Use**.
4. If the database accessed by your data source includes tables, select a table from the **Add Tables** dialog box.

   Microsoft Query displays the added table in the top half of the query designer.

   Note The Owner list is not available in this dialog box because the driver does not support owners. The Database list is not available because the driver does not support multiple databases in a data source.
5. Select fields for your query by dragging them from the table onto the lower half of the designer.
6. Close Microsoft Query.

   The data you selected is imported into your Microsoft Excel spreadsheet.

Creating Mailing Labels in Microsoft Word Using Visual FoxPro Data

You can use Visual FoxPro data in a Microsoft Word for Windows 95 or Windows 98 document. For example, you might want to create mailing labels from the customer information stored in a FoxPro table.

To create mailing labels

1. In Microsoft Word, create a new blank document.
2. From the **Tools** menu, choose **Mail Merge**.
3. In the Mail Merge Helper, choose **Create**, then select **Mailing Labels**.
4. Under **Main Document**, choose **Active Window**.
5. Under **Data Source**, choose **Get Data**, then select **Open Data Source**.
6. In the **Open Data Source** dialog box, choose **MS Query**.
7. In the **Select Data Source** dialog box, select a Visual FoxPro data source, then click **Use**.
8. If the database accessed by your data source includes tables, select a table from the **Add Tables** dialog box.

   Microsoft Query displays the added table in the top half of the query designer.
9. Select fields for your query by dragging them from the table onto the lower half of the designer.
10. From the **File** menu, choose **Return Data to Microsoft Word**.

    Microsoft Query closes, and the data you selected is available for use in your mail merge document.
11. Under **Main Document**, choose **Setup**.
12. In the **Label Options** dialog box, select the printer and label information you want, then click OK.
13. In the **Create Labels** dialog box, select the fields you want to print on the mailing labels, then click OK
14. .In the Mail Merge Helper, under **Merge the Data with the Document**, click **Merge**.
15. In the **Merge** dialog box, select the options you want, then click **Merge**.

Importing Visual FoxPro Data into Microsoft Access

You can import data stored in a Visual FoxPro database into a Microsoft Access database using the Import option.

To import Visual FoxPro data into a Microsoft Access database

1. Open a Microsoft Access database.
2. From the **File** menu, choose **Get External Data**, then **Import**.
3. In the **Import** dialog box select **ODBC Databases( )** in the Files of type list.
4. In the **SQL Data Sources** dialog box, select the Visual FoxPro data source that connects to the FoxPro data you want to query and click **OK**.
5. In the **Import Objects** dialog box, select one or more tables you want to import and click **OK**.

   The names of the Visual FoxPro tables you imported are displayed in the **Tables** tab of the Microsoft Access database.

You can now use Microsoft Access to manipulate the data in the imported Visual FoxPro tables. The data you import is a snapshot of the data stored in Visual FoxPro; changes you make to imported data are not sent back to the Visual FoxPro data source.

If you want changes you make in Microsoft Access to change the data on the Visual FoxPro data source, see Querying and Updating Visual FoxPro Data from Access.

Querying and Updating Visual FoxPro Data from Microsoft Access

You can query and update data stored in a Visual FoxPro database from a Microsoft Access database by using the Link Table option.

To link a Visual FoxPro database to a Microsoft Access database

1. Open a Microsoft Access database.
2. From the **Tables** tab, click **New**.
3. In the **New Table** dialog box, select Link Table and click **OK**.
4. In the **Link** dialog box, select **ODBC Databases( )** in the Files of type list.
5. In the **SQL Data Sources** dialog box, select the data source that connects to the Visual FoxPro data you want to query and click **OK**.
6. In the **Link Tables** dialog box, select the tables you want to query and update and click **OK**.

   The linked Visual FoxPro tables are displayed in the **Tables** tab of the Microsoft Access database.

You can now use Microsoft Access to query and update data in the linked Visual FoxPro tables. Changes you make to linked data are sent back to the Visual FoxPro data source.

If you don't want changes you make in Microsoft Access to affect the data on the Visual FoxPro data source, see Importing Visual FoxPro Data into Microsoft Access.

Using the Visual FoxPro ODBC Driver with your Visual Basic Application

Your Visual Basic application can communicate with Visual FoxPro data by creating a data control that connects to a Visual FoxPro data source.

To connect to Visual FoxPro data using the Data Control in Visual Basic

1. Create a data source called `test` that connects to the TasTrade sample database included in Visual FoxPro. The default Visual FoxPro installation places the TasTrade sample database in the location:
   `c:\vfp\samples\mainsamp\data\tastrade.dbc`
2. In Visual Basic , create a new form and place a text box and a Data control on it.
3. Change the Data control's Connect property to:
   `ODBC;DATABASE=tastrade;DSN=test`
4. Change the RecordsetType property to
   `2 - Snapshot`
5. Change the RecordSource property to:
   `customer`
6. Change the DataSource property for the text box to the default name for the Data control:
   `data1`
7. Change the text box's DataField property to
   `customer_id`
8. Run the form and use the Data control to skip through the customer id fields from the Visual FoxPro TasTrade sample database.

Using the Visual FoxPro ODBC Driver with your C or C++ Application

Example

Your application communicates with Visual FoxPro data by sending a **SQLExecute** or **SQLExecDirect** statement to Visual FoxPro. This statement can contain:

- SQL statements native to the Visual FoxPro language, such as the DROP TABLE command
- Supported ODBC SQL grammar.
- Non-SQL Visual FoxPro language such as supported SET commands.

For more information about SQL native to Visual FoxPro, see the Visual FoxPro documentation.

Using the Visual FoxPro ODBC Driver with your C or C++ Application

The following example uses the ODBC C API to retrieve data stored in the `last_name` field in the `employee` table in the Visual FoxPro sample database TasTrade. This database is provided with Visual FoxPro and is installed by default in the following location:

`c:\vfp\samples\mainsamp\data\tastrade.dbc`

The example displays one last name at a time, allowing you to click the OK button on the message box to see the next last name. It is assumed that a data source called tastrade has been set up to use the `tastrade.dbc` database.

Note Error checking should be performed on all ODBC API calls; this example excludes error checking for the sake of brevity.

```
//To the maximum extent permitted by law, Microsoft and //its suppliers
disclaim all warranties, either //expressed or implied, including, but not
limited to //implied warranties of merchantability and fitness for
//particular purpose, with regard to this example.

#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <stdlib.h>
#include <mbstring.h>

#define MAX_DATA 100
#define MYSQLSUCCESS(rc) ((rc==SQL_SUCCESS)||(rc==SQL_SUCCESS_WITH_INFO))

class direxec
{
   RETCODE rc;          // ODBC return code
   HENV henv;           // Environment
   HDBC hdbc;           // Connection handle
   HSTMT hstmt;         // Statement handle
   unsigned char szData[MAX_DATA]; // Returned data storage
   SDWORD cbData;       // Output length of data
   unsigned char chr_ds_name[SQL_MAX_DSN_LENGTH];          // Data source
name

public:
   direxec();                // Constructor
   void sqlconn();                 // Allocate env,stat,and connect
   void sqlexec(unsigned char *);  // Execute SQL statement
   void sqldisconn();              // Free pointers to env,stat,conn,
                           // and disconnect.
   void error_out();         // Displays errors.
};

// Constructor initializes the string chr_ds_name with the data source
// name.
direxec::direxec()
{
   _mbscpy(chr_ds_name,(const unsigned char *)"tastrade");
}

// Allocate environment handle, allocate connection handle,
```

```
// connect to data source, and allocate statement handle.
void direxec::sqlconn(void)
{
   SQLAllocEnv(&henv);
   SQLAllocConnect(henv,&hdbc);
   rc=SQLConnect(hdbc,chr_ds_name,SQL_NTS,NULL,0,NULL,0);

   // Deallocate handles, display error message, and exit.
   if (!MYSQLSUCCESS(rc))
   {
      SQLFreeEnv(henv);
      SQLFreeConnect(hdbc);
      error_out();
      exit(-1);
   }

   rc=SQLAllocStmt(hdbc,&hstmt);

}

// Execute SQL command with SQLExecDirect() ODBC API.
void direxec::sqlexec(unsigned char * cmdstr)
{
   rc=SQLExecDirect(hstmt,cmdstr,SQL_NTS);
   if (!MYSQLSUCCESS(rc))  //Error
   {
      error_out();
      // Deallocate handles and disconnect.
      SQLFreeStmt(hstmt,SQL_DROP);
      SQLDisconnect(hdbc);
      SQLFreeConnect(hdbc);
      SQLFreeEnv(henv);
      exit(-1);
   }
   else
   {
      for (rc=SQLFetch(hstmt); rc == SQL_SUCCESS; rc=SQLFetch(hstmt))
      {
         SQLGetData(hstmt,1,SQL_C_CHAR,szData,sizeof(szData),&cbData);
         // In this example, the data that is returned in a messagebox
         // for simplicity. However, normally the SQLBindCol() ODBC API
         // could be called to bind individual rows of data and assign
         // for a a rowset.  See Chapter 15: Returning Results in the
         // ODBC SDK for a completed details.
         MessageBox(NULL,(const char *)szData,"ODBC",MB_OK);
      }
   }
}

// Free the statement handle, disconnect, free the connection handle and
// free the environment handle.
void direxec::sqldisconn(void)
{
   SQLFreeStmt(hstmt,SQL_DROP);
   SQLDisconnect(hdbc);
   SQLFreeConnect(hdbc);
   SQLFreeEnv(henv);
```

```
}

// Display error message in a message box that has an OK button.
void direxec::error_out(void)
{
   unsigned char szSQLSTATE[10];
   SDWORD nErr;
   unsigned char msg[SQL_MAX_MESSAGE_LENGTH+1];
   SWORD cbmsg;

   while(SQLError(0,0,hstmt,szSQLSTATE,&nErr,msg,sizeof(msg),&cbmsg)==
SQL_SUCCESS)
   {
      wsprintf((char *)szData,"Error:\nSQLSTATE=%s,Native error=
%ld,msg='%s'",
         szSQLSTATE,nErr,msg);
      MessageBox(NULL,(const char *)szData,"ODBC Error",MB_OK);

   }

}

int WINAPI WinMain (HANDLE hInstance, HANDLE hPrevInstance,
               LPSTR lpszCmdLine, int nCmdShow)
{
   // Declare an instance of the direxec object.
   direxec x;

   // Allocate handles and connect.
   x.sqlconn();

   // Execute SQL command "SELECT last_name FROM employee"
   x.sqlexec((UCHAR FAR *)"SELECT last_name FROM employee");

   // Free handles and disconnect
   x.sqldisconn();

   // Return success code; example executed successfully.
   return (TRUE);
}
```

System Requirements

The system requirements for installation provide the minimum operating system and disk space needed to successfully install the driver. Once you've installed the driver, you can select the specific application software you want to use to access Visual FoxPro data.

Installation Requirements

To install the Microsoft Visual FoxPro ODBC Driver, you need:

- Windows NT 4 (or later), Windows 95, or Windows 98
- 2 MB disk space

For information about installing the driver, see Setting Up the Visual FoxPro ODBC Driver.

Accessing Visual FoxPro Data

To access Microsoft Visual FoxPro or FoxPro 2.x data, you must have the following:

- ODBC Client Software (automatically installed with the driver)
- Microsoft Visual FoxPro ODBC Driver
- Any of the following types of application software:
  - Microsoft Office application such as Microsoft Excel or Word
  - C or C++ language ODBC application
  - Visual Basic ODBC application
- Any of the following types of data:
  - Visual FoxPro database or a directory of free tables
  - FoxPro 2.0, 2.5, 2.6 table

The Visual FoxPro ODBC Driver supports double-byte character sets (DBCS). For more information, see International Support.

The driver does not support 16-bit Windows 3.1 applications.

Supported Versions of FoxPro

You can use the Visual FoxPro ODBC Driver to access data stored in FoxPro tables. The following versions of FoxPro data are supported:

- 2.0
- 2.5
- 2.6
- Visual FoxPro (all versions)

When you access data stored in Visual FoxPro, you can choose to connect to a database containing zero or more tables or to a directory of free tables.

For more information on connecting to a data source, see Adding a Visual FoxPro Data Source.

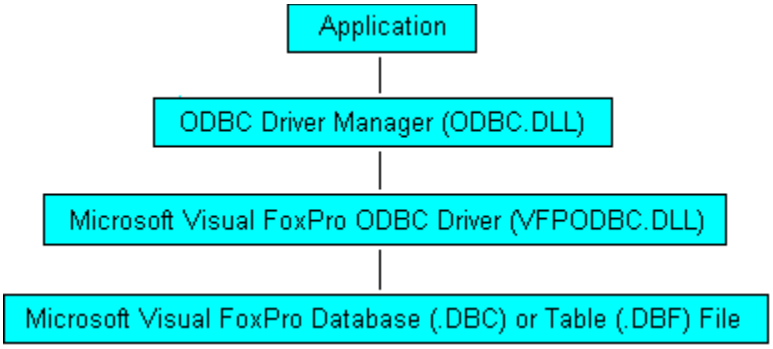
Overview of Driver Architecture

The Microsoft Visual FoxPro ODBC Driver is a 32-bit driver that enables you to open and query a Microsoft Visual FoxPro database or FoxPro tables through the Open Database Connectivity (ODBC) interface. You can access FoxPro data using:

- A Microsoft Office application, such as Excel or Microsoft Word, which uses Microsoft Query to communicate with ODBC.
- An application written in Visual C++ or C that uses the ODBC SDK API.
- An application written in Visual Basic or Visual Basic for Applications.

In each case, the request for information uses the ODBC API. The ODBC Driver Manager works with the Visual FoxPro ODBC Driver to open and retrieve data from FoxPro tables and databases.

The architecture is represented in the following diagram:



Bookmark Support

The Visual FoxPro ODBC Driver supports simple bookmarks. When you call **SQLGetInfo** with the SQL_BOOKMARK_PERSISTENCE InfoType, the return value is SQL_BP_SCROLL.

For information about bookmarks, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Supported Concurrency Model

The Visual FoxPro ODBC Driver supports *read-only* concurrency. Your application can call **SQLSetStmtOption** with a SQL_CONCURRENCY option of SQL_CONCUR_READ_ONLY.

For more information about concurrency and **SQLSetStmtOption**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

read-only concurrency

The cursor cannot be updated.

row versioning

Essentially timestamp support, in which row versions are compared at update time.

Supported Cursor Model

The Visual FoxPro ODBC Driver supports both *block* (*rowset*) and *static* cursors. Static cursors are supported for any driver that conforms to Level 1 ODBC compliance. The driver does not support dynamic, keyset-driven, or mixed (keyset and dynamic) cursors.

Your application can call **SQLSetStmtOption** with a SQL_CURSOR_TYPE option of SQL_CURSOR_FORWARD_ONLY (block cursor) or SQL_CURSOR_STATIC (static cursor).

Note If you call **SQLSetStmtOption** with a SQL_CURSOR_TYPE option other than SQL_CURSOR_FORWARD_ONLY or SQL_CURSOR_STATIC, the function returns SQL_SUCCESS_WITH_INFO with a SQLSTATE of 01S02 (Option value changed). The driver sets all unsupported cursor modes to SQL_CURSOR_STATIC.

For more information on cursor types and on **SQLSetStmtOption**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

block cursor

A forward-scrolling, read-only result set returned to the client, who is responsible for maintaining storage for the data.

static cursor

A snapshot of a set of data defined by the query. Static cursors do not reflect real-time changes by other users of the underlying data. The cursor's memory buffer is maintained by the ODBC cursor library, which allows forward and backward scrolling.

rowset

Blocks of data stored in a cursor, representing rows retrieved from a data source.

Data Types

The list of data types supported by the driver are presented through the ODBC API and in Microsoft Query.

Data Types in C Applications

You can obtain a list of data types supported by the Visual FoxPro ODBC Driver by using the **SQLGetTypeInfo** function in C or C++ applications.

Data Types in Applications Using Microsoft Query

If your application uses Microsoft Query to create a new table on a Visual FoxPro data source, Microsoft Query displays the **New Table Definition** dialog box. Under **Field Description**, the **Type** box lists Visual FoxPro field data types, represented by single characters.

International Support

The Microsoft Visual FoxPro ODBC Driver supports:

- Double-byte character sets (DBCS)
- Multiple collating sequences

A collating sequence defines the sort order for data stored in a Visual FoxPro table or database. By default the driver is configured to use the collating sequences that support the language version of your operating system.

For a list of supported collating sequences, see SET COLLATE.

locale

The set of information that corresponds to a given language and country. A locale indicates specific settings such as decimal separators, date and time formats, and character-sorting order.

sort order

Sort orders incorporate the sorting rules of different locales, allowing you to sort data in those languages correctly. In Visual FoxPro, the current sort order determines the results of character expression comparisons and the order in which the records appear in indexed or sorted tables.

ODBC Visual FoxPro Setup Dialog Box

Enables you to add or change a Visual FoxPro data source.

Dialog Box Options

Data Source Name Enter the name you want to use for the data source.

Description Enter a description for the data source.

Database type Lets you choose the type of database you want your data source to connect to.

Visual FoxPro database (.DBC) Specifies that the data source connects to a Visual FoxPro database (.DBC file) and all the tables and local views in the database.

Free Table directory Specifies that the data source connects to a directory of free tables. Any database tables in the same directory are ignored by ODBC catalog functions such as **SQLColumns** or **SQLTables**. Database tables can be accessed by using SQL SELECT statements sent through **SQLExecute** and **SQLExecDirect**.

Path Displays the path and name for the database or the directory of free tables to which the data source connects

Browse Enables you to search your system and network for the database or directory to which you want to connect the data source.

Options Expands the dialog box so you can see and sets Visual FoxPro ODBC Driver options.

Driver

Collating sequence The sequence in which fields are sorted. The default sequences reflect the sequences supported by your language version of the operating system. For a list of supported collating sequences, see SET COLLATE.

Exclusive When this checkbox is checked, the driver opens the Visual FoxPro database exclusively when you access data using the data source. Other users cannot access the database or the tables in the database while the database is opened exclusively. Tables within the exclusively opened database are opened as SHARED. To open a table exclusively, use the SET EXCLUSIVE command. This checkbox is disabled when the Database type is set to **Free Table directory**.

Fetch data in background Determines whether records will be fetched in the background (progressive fetching) or your application will wait until all records in the result set are fetched.

Supported ODBC SQL Grammar

The Microsoft Visual FoxPro ODBC Driver supports:

- All SQL statements and clauses in the ODBC minimum SQL grammar
- An additional SQL statement from the ODBC core SQL grammar.

The following table lists the items supported by the driver by ODBC SQL Grammar level.

Level	Elements	Item
Minimum	Data Definition Language (DDL)	CREATE TABLE and DROP TABLE
	Data Manipulation Language (DML)	SELECT, INSERT, UPDATE, and DELETE
	Expressions	Simple (such as A>B+C)
	Data Types	CHAR, VARCHAR, or LONG VARCHAR

In addition to the supported ODBC SQL grammar, the Visual FoxPro ODBC Driver supports the complete native Visual FoxPro language syntax for these Visual FoxPro commands:

ALTER TABLE

CREATE TABLE

DELETE

DELETE TAG

DROP TABLE

INDEX

INSERT

SELECT

UPDATE

Registry Entries

When you install the Visual FoxPro ODBC Driver, the installation program updates your system's registry, HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI, to add a new key called Microsoft Visual FoxPro Driver. Under that key, the following values are added:

Value Name	Value Type	Value
APILevel	REG_SZ	"1"
ConnectFunctions	REG_SZ	"YYN"
Driver	REG_SZ	System path to the VFPODBC.DLL file
DriverODBCVer	REG_SZ	"02.50"
FileExtns	REG_SZ	"*.dbf,*.cdx,*.fpt"
FileUsage	REG_SZ	"1"
Setup	REG_SZ	System path to the VFPODBC.DLL file
SQLLevel	REG_SZ	"0"

The installation program also adds the key "Visual FoxPro Files", representing the default Visual FoxPro driver, to your system's HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI key. Under this key, the installation program adds the following values:

Value Name	Value Type	Value
Driver	REG_SZ	System path to the VFPODBC.DLL file

Each time you add a Visual FoxPro ODBC data source to your ODBC configuration, a new key is added for that data source name. The values for the data source correspond to values you set in the **ODBC Visual FoxPro Setup** dialog box, as listed in the following table:

Value Name	Value Type	Value
Collate	REG_SQ	Any supported collating sequence
Description	REG_SZ	User description of data source
Driver		System path to the VFPODBC.DLL file
Exclusive		Yes or No
BackgroundFetch		Yes or No
SourceDB	REG_SZ	Path to .DBC file
SourceType	REG_SZ	"DBC" or "DBF"

You should not access this information directly; any administration of the registry is handled by the ODBC administrator as you add, modify or delete a data source.

You can use some of these keywords and values entered when you use the **SQLDriverConnect** ODBC API function.

Supported Scalar Functions

The Visual FoxPro ODBC Driver supports three types of scalar functions as defined in the ODBC SQL grammar:

- String functions
- Numeric functions
- Time and date functions

String Functions

The following table lists ODBC string manipulation functions supported by the Visual FoxPro ODBC Driver; when the Visual FoxPro grammar for the same function differs from the ODBC syntax, the Visual FoxPro equivalent is listed.

ODBC Grammar	**Visual FoxPro Grammar**
ASCII (*string_exp*)	ASC *(string_exp*)
CHAR *(code*)	CHR (*string_exp*)
CONCAT (*string_exp1, string_exp2*)	*string_exp1* + *string_exp2*
DIFFERENCE (*string_exp1, string_exp2*)	
INSERT (*string_exp1, start, length, string_exp2*)	STUFF (*string_exp1, start, length, string_exp2*)
LCASE (*string_exp*)	LOWER (*string_exp*)
LEFT (*string_exp, count*)	
LENGTH (*string_exp*)	LEN (*string_exp*)
LTRIM (*string_exp*)	
REPEAT (*string_exp, count*)	REPLICATE (*string_exp, count*)
REPLACE (*string_exp1, string_exp2, string_exp3*)	STRTRAN (*string_exp1, string_exp2, string_exp3*)
RIGHT (*string_exp, count*)	
RTRIM (*string_exp*)	
SOUNDEX (*string_exp*)	
SPACE (*count*)	
SUBSTRING (*string_exp, start, length*)	SUBSTR (*string_exp, start, length*)
UCASE (*string_exp*)	UPPER (*string_exp*)

Numeric Functions

The following table describes ODBC numeric functions supported by the Visual FoxPro ODBC Driver; when the Visual FoxPro grammar for the same function differs from the ODBC syntax, the Visual FoxPro equivalent is listed.

ODBC Grammar	**Visual FoxPro Grammar**
ABS (*numeric_exp*)	
ACOS (*float_exp*)	
ASIN (*float_exp*)	
ATAN (*float_exp*)	
ATAN2 (*float_exp1, float_exp2*)	ATN2(*float_exp1, float_exp2*)
CEILING (*numeric_exp*)	
COS (*float_exp*)	
COT (*float_exp*)	
DEGREES (*numeric_exp*)	RTOD (*numeric_exp*)
EXP (*float_exp*)	
FLOOR (*numeric_exp*)	
LOG (*float_exp*)	
LOG10 (*float_exp*)	
MOD (*integer_exp1, integer_exp2*)	
PI ( )	
RADIANS (*numeric_exp*)	DTOR (*numeric_exp*)
RAND (*[integer_exp]*)	
ROUND (*numeric_exp, integer_exp*)	
SIGN (*numeric_exp*)	
SIN (*float_exp*)	
SQRT (*float_exp*)	
TAN (*float_exp*)	

The following numeric functions are not supported:

POWER (*numeric_exp, integer_exp*)

TRUNCATE (*numeric_exp, integer_exp*)

Time and Date Functions

The following table lists ODBC time and date functions supported by the Visual FoxPro ODBC Driver; when the Visual FoxPro grammar for the same function differs from the ODBC syntax, the Visual FoxPro equivalent is listed.

ODBC Grammar	Visual FoxPro Grammar
CURDATE( )	DATE( )
CURTIME( )	TIME( )
DAYNAME(*date_exp*)	CDOW(*date_exp*)
DAYOFMONTH(*date_exp*)	DAY( )
HOUR(*time_exp*)	
MINUTE(*time_exp*)	
MONTH(*time_exp*)	
MONTHNAME(*date_exp*)	CMONTH(*date_exp*)
NOW( )	DATETIME( )
SECOND(*time_exp*)	SEC(*time_exp*)
WEEK(*date_exp*)	
YEAR(*date_exp*)	

The following time and date functions are not supported:

DAYOFYEAR (*date_exp*)

QUARTER (*date_exp*)

TIMESTAMPADD (*interval, integer_exp, timestamp_exp*)

TIMESTAMPDIFF (*interval, timestamp_exp1, timestamp_exp2*)

ODBC Escape Sequences

The driver also supports the ODBC escape sequence for date and timestamp data. The escape clause syntax is:

--(*vendor(Microsoft),product(ODBC) d *'value'* ***)**—
--(*vendor(Microsoft),product(ODBC) ts ' *'value'* ***)**—

In this syntax, **d** indicates *value* is a date in the "yyyy-mm-dd" format and **ts** indicates *value* is a timestamp in the "yyyy-mm-dd hh:mm:ss[.f…]" format. The shorthand syntax for date and timestamp data is:

{**d** *'value'*}
{**ts** *'value'*}

For example, each of the following statements update the ALLTYPES table using the date and timestamp shorthand syntax in a supported SQL UPDATE command:

```
UPDATE alltypes
   SET DAT_COL={d'1968-04-28'}
   WHERE KEY=111

UPDATE alltypes
   SET DTI_COL={ts'1968-04-28 12:00:00'}
   WHERE KEY=111
```

For more information about escape sequences, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Supported SET Commands

Your application can send the following Visual FoxPro SET commands to a data source:

SET ANSI

SET BLOCKSIZE

SET COLLATE

SET DELETED

SET EXACT

SET EXCLUSIVE

SET NULL

SET PATH

SET REPROCESS

SET UNIQUE

Thread Support

The Visual FoxPro ODBC Driver is thread-safe. Access to environment handles (*henv*), connection handles (*hdbc*), and statement handles (*hstmt*) are wrapped in appropriate semaphores to prevent other processes from accessing and potentially altering the driver's internal data structures.

In a multithreaded application, you can cancel a function that is running synchronously on an *hstmt* by calling **SQLCancel** on a separate thread.

The driver uses a separate thread to fetch data when you use progressive fetching. To employ progressive fetching for a data source, check the **Fetch data in background** checkbox on the ODBC Visual FoxPro Setup dialog box, or use the BackgroundFetch attribute keyword in your connection string. For information on connection string attribute keywords, see Using Connection Strings.

For more information about threads and **SQLCancel**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Troubleshooting

The following sections discuss how to improve performance and solve problems you might encounter while using the Visual FoxPro ODBC Driver.

Accessing Parameterized Views

You can't access parameterized views in a Visual FoxPro database using the driver. A parameterized view creates a WHERE clause in the view's SQL SELECT statement that limits the records downloaded to only those records that meet the conditions of the WHERE clause built using the value supplied for the parameter. Because the driver doesn't support passing parameters to the view, attempts to access a parameterized view will fail.

The parameter value can be:

- Supplied at run time.
- Passed programmatically to the view.

Accessing Remote Views

You can't access remote views in a Visual FoxPro database using the driver. Remote views are views that access either non-FoxPro data or a combination of FoxPro and non-FoxPro data. To access remote views, use Visual FoxPro.

Deleting Records

You can mark records for deletion using the driver, but you can't permanently remove records from the database. To permanently remove records from a table, use Visual FoxPro.

Increasing Performance Using Background Fetching

You can improve performance on large fetches by using the background fetching feature of the driver. Background fetching uses a separate thread to fetch data requested from a specific data source.

You can employ background fetching for a data source in one of two ways:

- Check the **Fetch data in background** checkbox on the ODBC Visual FoxPro Setup dialog box.
- Use the BackgroundFetch attribute keyword in your connection string.

For information on connection string attribute keywords, see Using Connection Strings.

Updating MultiTiered Views

A multitiered view is a view based on one or more views rather than on a base table. When you update data in a multitiered view, the updates go down only one level, to the view on which the top-level view is based; base tables are not updated.

Using Data Definition Language (DDL) in Stored Procedures

You can't use DDL, such as CREATE TABLE, or ALTER TABLE in Visual FoxPro stored procedures.

For information on language you can use in stored procedures, see Visual FoxPro Language Support in Rules, Triggers, Default Values, and Stored Procedures.

Using Positioned Updates

The driver doesn't support positioned updates. Use the SQL WHERE clause to identify the rows you want to update.

Using the SET ANSI Command

If you're a Visual FoxPro developer, you should be aware that the default setting for SET ANSI is ON for the driver, in contrast to a default setting of OFF for Visual FoxPro. The default ON setting for SET ANSI allows Visual FoxPro data sources to behave consistently with other ODBC data sources that typically perform exact comparisons. You can change the default setting. For more information, see SET ANSI.

Error Messages Overview

When an error occurs, the Visual FoxPro driver returns the following information:

- The native error number and error message text.
- The SQLSTATE (an ODBC error code) and error message text.

You access this error information by calling **SQLError**.

Native Errors

For errors that occur in the data source, the Visual FoxPro driver returns the native error number and error message text. For a list of native error numbers, see Visual FoxPro ODBC Driver Native Error Messages.

SQLSTATE (ODBC Error Codes)

For errors that are detected and returned by the Visual FoxPro driver, the driver maps the returned native error number to the appropriate SQLSTATE. If a native error number does not have an ODBC error code to map to, the Visual FoxPro driver returns SQLSTATE S1000 (General error).

For a list of SQLSTATE values generated by the Visual FoxPro ODBC Driver for corresponding Visual FoxPro errors, see ODBC Error Codes.

Syntax

Error messages have the following format:

[*vendor***][***ODBC_component***]***error_message*

The prefixes in brackets ([ ]) identify the source of the error as defined in the following table.

Data Source	Prefix	Value
Driver Manager	[vendor]	[Microsoft]
	[ODBC_component]	[ODBC Driver Manager]
	[data_source]	N/A
Visual FoxPro driver	vendor]	[Microsoft]
	[ODBC_component]	[ODBC Visual FoxPro driver]
	[data_source]	N/A

For example, if the Visual FoxPro ODBC Driver could not find the file EMPLOYEE.DBF, it might return the following error message:

“[Microsoft][ODBC Visual FoxPro Driver]File ‘employee.dbf’ does not exist”

Visual FoxPro ODBC Driver Native Error Messages

The following table lists error messages native to the Visual FoxPro ODBC Driver.

001

1	Feature is not available.
2	Input/output operation failure.
3	Free handle is not found.
5	Use of unallocated handle.
14	???????
97	???????
99	Procedure canceled.

100

100	Too many files open.
101	Cannot open file.
102	Cannot create file.
105	Error writing to file.
107	Invalid key length.
109	Record is out of range.
110	Record is not in index.
111	Invalid file descriptor.
113	File is not open.
114	Not enough disk space for *value*.
115	Invalid operation for the cursor.
118	Index file does not match table.
119	No table is open.
120	File does not exist.
121	File already exists.
122	Table has no index order set.
123	Not a table.
125	Index expression exceeds maximum length.
127	You must use a logical expression with a FOR or WHILE clause.
128	Not a numeric expression.
129	Variable is not found.
132	File is in use.
133	Index does not match the table. Delete the index file and re-create the index.
135	End of file encountered.
136	Beginning of file encountered.
137	Alias is not found.
139	You must use a logical expression with FILTER.

142	Cyclic relation.
143	No fields were found to copy.
144	The LOCATE command must be issued before the CONTINUE command.
145	Must be a character or numeric key field.
146	Cannot write to a read-only file.
147	Target table is already engaged in a relation.
148	Expression has been re-entered while the filter is executing.
149	Not enough memory for buffer.
150	Not enough memory for file map.
155	Invalid buffdirty call.
156	Duplicate field names.
158	No fields found to process.
159	Numeric overflow. Data was lost.
162	Procedure '*value*' is not found.
165	*value* is not related to the current work area.
170	Variable '*value*' is not found.
171	Cannot open file *value*.
173	File '*value*' does not exist.
174	'*value*' is not a memory variable.
175	'*value*' is not a file variable.
176	'*value*' is not an array.
177	Alias '*value*' is not found.
180	File was not placed in memory using the LOAD command.
182	There is not enough memory to complete this operation.

200

200	Syntax error.
201	Too many names used.
202	Program is too large.
203	Too many memory variables.
205	Nesting error.
206	Recursive macro definition.
209	Line is too long.
210	Allowed DO nesting level exceeded.
211	An IF | ELSE | ENDIF statement is missing.
212	Structure nesting is too deep.
213	There is a missing keyword in the FOR...ENDFOR or DO CASE...ENDCASE command structure.
219	Command contains unrecognized phrase/keyword.
221	Command is missing required clause.
222	Unrecognized command verb.
224	Invalid subscript reference.

227	Missing expression.
228	Table number is invalid.
229	Too few arguments.
230	Too many arguments.
233	Statement is not allowed in interactive mode.
234	Subscript is outside defined range.
236	Suspend program before using RESUME.
238	No PARAMETER statement is found.
239	Must specify additional parameters.
240	Not a character expression.
250	Too many PROCEDURE commands are in effect.
252	Compiled code for this line is too long.
257	Key string is too long.
291	Expression used with ASIN() is out of range.
292	Cannot use 0 or negative as the argument for LOG10().
293	Expression used with ACOS() is out of range.
294	FOXUSER.DBF file is invalid.
295	Invalid path or file name.
296	Error reading the resource.
297	Command is allowed only in interactive mode.

300

301	Operator/operand type mismatch.
302	Data type mismatch.
305	Expression evaluated to an illegal value.
307	Cannot divide by 0.
308	Insufficient stack space.
337	Cannot nest the PRINTJOB command.

400

406	Printer is not ready.
407	Invalid argument used with the SET function.
410	Unable to create temporary work files.
423	Error creating the OLE object.
424	Error copying the OLE object to the Clipboard.
462	*value* internal consistency error.
465	SQL pass-through internal consistency error.
466	Connection handle is invalid.
467	Property is invalid for local cursors.
468	Property is invalid for table cursors.
469	Property value is out of bounds.
470	Incorrect property name.
471	Incorrect column format.
473	Environment-level property is invalid.

474	Invalid call issued while executing a SQLEXEC() sequence.
479	Invalid update column name *value*\.
489	General fields cannot be used in the WHERE condition of an update statement. Change the WhereType property of the view.
491	No update tables are specified. Use the Tables property of the cursor.
492	No key columns are specified for the update table \ *value*\. Use the KeyFieldList property of the cursor.
493	SQL parameter is missing.
494	View definition has been changed.
495	Warning: The key defined by the KeyField property for table *value* is not unique
498	SQL SELECT statement is invalid.
499	SQL parameter *value* is invalid.

500

502	Cannot write to the record because it is in use.
503	File cannot be locked.
508	Error initializing OLE.
520	No database is open or set as the current database.
522	Connectivity internal consistency error.
523	Execution was canceled by the user.
525	Function is not supported on remote tables.
526	Connectivity error: *value*
527	Cannot load ODBC library, ODBC32.DLL.
528	ODBC entry point missing, *value*.
530	Fetching canceled; remote table is closed.
532	Type conversion is not supported.
533	This property is read-only.
536	Function is not supported on native tables.
538	A stored procedure is executing.
540	Session number is invalid.
541	Connection *value* is busy.
542	Base table fields have been changed and no longer match view fields. View field properties cannot be set.
543	Type conversion required by the DataType property for field '*value*' is invalid.
544	DataType property for field '*value*' is invalid.
545	Table buffer for alias *value*\ contains uncommitted changes.
546	Cannot close table during execution of table-bound expression.
547	Cannot insert an empty row from a view into its base table(s).
548	Table *value* has one or more non-structural indexes

	open. Please close them and retry the Begin Transaction
549	Data session #*value* cannot be released with open transaction(s).
550	.DBC internal consistency error.
557	The database must be opened exclusively.
559	Property is not found.
560	Property value is invalid.
561	Database is invalid. Please validate.
562	Cannot find object *value* in the database.
563	Cannot find view *value* in the current database.
566	Cannot issue the PACK command on a database while its tables are in use.
567	Primary key property is invalid; please validate database.
570	Database is read-only.
571	The name *value* is already used for another
575	Object name is invalid.
577	Table *value* is referenced in a relation.
578	Invalid database table name.
579	Command cannot be issued on a table with cursors in table buffering mode.
580	Feature is not supported for non-.DBC tables.
581	Field *value* does not accept null *value*.
583	Record validation rule is violated.
585	Update conflict. Use TABLEUPDATE() with the lForce parameter to commit the update or TABLEREVERT() to roll back the update.
586	Function requires row or table buffering mode.
587	Illegal nested OLDVAL() or CURVAL().
589	Table or row buffering requires that SET MULTILOCKS is set to ON.
590	BEGIN TRANSACTION command failed. Nesting level is too deep.
591	END TRANSACTION command cannot be issued without a corresponding BEGIN TRANSACTION command.
592	ROLLBACK command cannot be issued without a corresponding BEGIN TRANSACTION command.
593	Command cannot be issued within a transaction.
594	Illegal to attempt a file lock in a transaction after taking prior record locks.
596	Table buffering is not enabled.
597	Views require either DB_BUFOPTROW or DB_BUFOPTTABLE.
598	Rule and trigger code must balance transaction usage.
599	Data session #*value* was forced to ROLLBACK all transactions to avoid deadlock.

600

601	Alias name is already in use.
602	Operation is invalid for a Memo, General, or Picture field.
612	No such menu or menu item is defined.
618	Menu has not been defined with DEFINE MENU.
624	Menu title has not been defined with DEFINE PAD.
625	Menu has not been defined with DEFINE POPUP.
631	Array dimensions are invalid.
637	File must be opened exclusively to convert the Memo file.
638	Field must be a Memo field.
649	No previous PRINTJOB command to correspond to this command.
651	CANCEL or SUSPEND is not allowed.
659	The table has memo fields that cannot be converted while open read-only.
683	Index tag is not found.

700

700	Record is in use by another user.
701	File must be opened exclusively.
702	File is in use by another user.
703	Record is not locked.
705	File access is denied.
706	Cannot sort .IDX files in descending order.
707	Structural .CDX file is not found.
708	File is open in another work area.
712	Field name is a duplicate or invalid.
714	Window '*value*' has not been defined.
718	File is read-only.
722	Preprocessor expression is invalid.
734	Property *value* is not found.
737	*value* is a method, event, or object.
738	Property *value* is not a method or event.
740	*value* is a read-only property.
748	This file is incompatible with the current version of Visual FoxPro. .
750	File was created in a later version of Visual FoxPro than the current version.
763	Property *value* already exists.
773	Database object type is invalid.
784	This object is derived from a base class and does not have a parent class.

800

802	SQL: Cannot locate table.
872	Too many columns.
879	No primary key.
884	Uniqueness of index *value* is violated.
885	Only structural tags can be defined as candidate.
886	Index does not accept NULL.
887	Illegal recursion in rule evaluation.
888	Tag name is too long.

900

901	Function argument value, type, or count is invalid.
902	Expression evaluator failed.
903	String is too long to fit.
904	** or ^ domain error.
905	LOG(): Zero or negative used as argument.
906	SQRT() argument cannot be negative.
912	Operation is invalid for a General field.
914	Code page number is invalid.
915	Collating sequence '*value*' is not found.
918	File name is too long.
922	Volume does not exist.
923	Object *value* is not found.
924	*value* is not an object.
925	Unknown member *value*.
928	Statement is only valid within a class definition.
929	*value* can only be used within a method.
930	Cannot redefine *value*.
931	Statement is not in a procedure.
934	Statement is only valid within a method.
935	The current object does not inherit from class *value*.
937	Procedure file '*value*' is not found.
938	Object is not contained in a *value*.
939	WITH/ENDWITH mismatch.
940	Expression is not valid outside of WITH/ENDWITH.
941	Error code is invalid.
942	Objects cannot be assigned to arrays.
943	Member *value* does not evaluate to an object.
945	The current object has been released.
947	Expression is too complex.
951	Cannot clear the object in use.
955	WIN.INI/registry is corrupted.
957	Error accessing printer spooler.
959	Invalid coordinates.
960	Illegal redefinition of variable *value*.

971	Cannot compile until the current COMPILE command has completed.
972	Array *value* is in use.
974	Arrays cannot be assigned to array elements.
976	Cannot resolve backlink.
988	Currency value is out of range.
990	Cancel.
999	Function is not implemented.

ODBC Error Codes

The following table lists Visual FoxPro error codes mapped to ODBC Error Code SQLSTATE values. The mapped SQLSTATE values come from **SQLExecDirect** and **SQLPrepare**. No other SQLSTATE values from other ODBC API are mapped because **SQLExecDirect** and **SQLPrepare** are the only functions that access the Visual FoxPro engine.

For more information on ODBC error codes, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSTATE	Visual FoxPro Error Code
S1001	149 150 182 202 308
1004	159
37000	132 200 219 221 222 227 229 230 498 499 713 901
22005	301 302
22012	307
23000	581 583 884 886 988
S0001	121 571
S0002	173 120 123 295 562 563 802
S0012	683
S0021	156 712
S0022	158

	806
S1000	100
	101
	102
	105
	107
	109
	110
	111
	113
	114
	115
	118
	119
	125
	133
	135
	136
	137
	145
	146
	171
	173
	177
	201
	205
	239
	240
	252
	257
	296
	305
	407
	410
	462
	502
	503
	520
	538
	550
	561
	567
	570
	575
	578
	580
	585
	602
	702
	705
	707
	708
	718
	750
	872

879
887
888
912
914
915
918
922
923
947
976
999

Visual FoxPro Language Support in Rules, Triggers, Default Values, and Stored Procedures

You cannot create Visual FoxPro rules, triggers, default values or stored procedures using the Visual FoxPro ODBC Driver. However, your application might interact with existing rules, triggers, default values or stored procedures as it inserts, updates, or deletes Visual FoxPro data stored in a database.

The following table lists the Visual FoxPro commands and functions supported by the Visual FoxPro ODBC Driver when the commands or functions exist in rules, triggers, default values or stored procedures.

If your application interacts with data whose rules, triggers, default values or stored procedures call any other Visual FoxPro commands or functions, the driver generates an error. See Unsupported Visual FoxPro Commands and Functions for a list of commands and functions not supported by the driver.

Tip If you want to insert conditional code into your rules, triggers or stored procedures that determines the commands to execute when called by the driver, you can use the VERSION( ) function. The VERSION( ) function returns "Visual FoxPro ODBC Driver <*version*>" when called by the driver.

Visual FoxPro Commands and Functions Supported in Rules, Triggers, Default Values and Stored Procedures

$ Operator	% Operator	& Command
&& Command	* Command	= Command

A

ABS( ) Function	ACOPY( ) Function	ACOS( ) Function
ADATABASES( ) Function	ADBOBJECTS( ) Function	ADD TABLE Command
ADEL( ) Function	AELEMENT( ) Function	AERROR( ) Function
AFIELDS( ) Function	AINS( ) Function	ALEN( ) Function
ALIAS( ) Function	ALLTRIM( ) Function	ALTER TABLE - SQL Command
AND Operator	ANSITOOEM( ) Function	APPEND Command
APPEND FROM ARRAY Command	APPEND FROM Command	APPEND GENERAL Command
APPEND MEMO Command	APPEND PROCEDURES Command	ASC( ) Function
ASCAN( ) Function	ASIN( ) Function	ASORT( ) Function
ASUBSCRIPT( ) Function	AT( ) Function	AT_C( ) Function
ATAN( ) Function	ATC( ) Function	ATCC( ) Function
ATCLINE( ) Function	ATLINE( ) Function	ATN2( ) Function
AUSED( ) Function	AVERAGE Command	

B

BEGIN TRANSACTION Command
BETWEEN( ) Function
BITAND( ) Function
BITCLEAR( ) Function
BITLSHIFT( ) Function
BITNOT( ) Function
BITOR( ) Function
BITRSHIFT( ) Function
BITSET( ) Function
BITTEST( ) Function
BITXOR( ) Function
BLANK Command
BOF( ) Function

C

CALCULATE Command
CANDIDATE( ) Function
CD Command
CDOW( ) Function
CDX( ) Function
CEILING( ) Function
CHDIR Command
CHR( ) Function
CHRTRAN( ) Function
CHRTRANC( ) Function
CLOSE Commands
CMONTH( ) Function
CONTINUE Command
COPY INDEXES Command
COPY MEMO Command
COPY PROCEDURES Command
COPY STRUCTURE Command
COPY STRUCTURE EXTENDED Command
COPY TAG Command
COPY TO ARRAY Command
COPY TO Command
COS( ) Function
COUNT Command
CPCONVERT( ) Function
CPCURRENT( ) Function
CPDBF( ) Function
CTOD( ) Function
CTOT( ) Function
CURDIR( ) Function
CURSORGETPROP( ) Function
CURSORSETPROP( ) Function
CURVAL( ) Function

D

DATE( ) Function
DATETIME( ) Function
DAY( ) Function
DBC( ) Function
DBF( ) Function
DBGETPROP( ) Function
DBUSED( ) Function
DECLARE Command
DELETE - SQL Command
DELETE Command
DELETE FILE Command
DELETE TAG Command
DELETED( ) Function
DESCENDING( ) Function
DIFFERENCE( ) Function
DIMENSION Command
DISKSPACE( ) Function
DMY( ) Function
DO CASE ... ENDCASE Command
DO Command
DO WHILE ... ENDDO Command
DOW( ) Function
DTOC( ) Function
DTOR( ) Function
DTOS( ) Function
DTOT( ) Function

E

EMPTY( ) Function
END TRANSACTION Command
EOF( ) Function

ERASE Command
ERROR Command
ERROR( ) Function
EVALUATE( ) Function
EXIT Command
EXP( ) Function

F

FCHSIZE( ) Function
FCLOSE( ) Function
FCOUNT( ) Function
FCREATE( ) Function
FDATE( ) Function
FEOF( ) Function
FERROR( ) Function
FFLUSH( ) Function
FGETS( ) Function
FIELD( ) Function
FILE( ) Function
FILTER( ) Function
FLDLIST( ) Function
FLOCK( ) Function
FLOOR( ) Function
FLUSH Command
FOPEN( ) Function
FOR ... ENDFOR Command
FOR( ) Function
FOUND( ) Function
FPUTS( ) Function
FREAD( ) Function
FREE TABLE Command
FSEEK( ) Function
FSIZE( ) Function
FTIME( ) Function
FULLPATH( ) Function
FUNCTION Command
FV( ) Function
FWRITE( ) Function

G

GATHER Command
GETCP( ) Function
GETENV( ) Function
GETEXPR Command
GETFLDSTATE( ) Function
GETNEXTMODIFIED( ) Function
GO/GOTO Command
GOMONTH( ) Function

H

HEADER( ) Function
HOME( ) Function
HOUR( ) Function

I

IDXCOLLATE( ) Function
IF ... ENDIF Command
IIF( ) Function
INDBC( ) Function
INDEX Command
INLIST( ) Function
INSERT-SQL Command
INT( ) Function
ISALPHA( ) Function
ISBLANK( ) Function
ISDIGIT( ) Function
ISEXCLUSIVE( ) Function
ISLEADBYTE( ) Function
ISLOWER( ) Function
ISNULL( ) Function
ISREADONLY( ) Function
ISUPPER( ) Function

J

K

KEY( ) Function
KEYMATCH( ) Function

L

LEFT( ) Function
LEFTC( ) Function
LEN( ) Function
LENC( ) Function
LIKE( ) Function
LIKEC( ) Function

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Unsupported FoxPro Commands and Functions

The following table lists FoxPro commands and functions that are not supported by the Visual FoxPro ODBC Driver, but are supported by Microsoft Visual FoxPro.

If your application interacts with data whose rules, triggers, default values or stored procedures call these Visual FoxPro commands or functions, the driver generates an error.

Unsupported Visual FoxPro Commands and Functions

#DEFINE ... #UNDEF	#IF ... #ENDIF Preprocessor Directive	#IFDEF | #IFNDEF
#INCLUDE Preprocessor Directive	:: Scope Resolution Operator	! Command (see RUN | ! Command)
? | ?? Command	??? Command	\ | \\ Command
@ ... BOX Command	@ ... CLASS Command	@ ... CLEAR Command
@ ... EDIT - Edit Boxes Command	@ ... FILL Command	@ ... GET
@ ... MENU Command	@ ... PROMPT Command	@ ... SAY Command
@ ... SCROLL Command	@ ... TO Command	

A

ACCEPT Command	ACLASS( ) Function	ACTIVATE MENU Command
ACTIVATE POPUP Command	ACTIVATE SCREEN Command	ACTIVATE WINDOW Command
ActivateCell Method	ADD CLASS Command	ADIR( ) Function
AFONT( ) Function	AINSTANCE( ) Function	_ALIGNMENT System Memory Variable
AMEMBERS( ) Function	APRINTERS( ) Function	ASELOBJ( ) Function
ASSIST Command		

B

BAR( ) Function	BARCOUNT( ) Function	BARPROMPT( ) Function
_BEAUTIFY System Memory Variable	_BOX System Memory Variable	BROWSE Command
_BROWSER System Memory Variable	BUILD APP Command	BUILD EXE Command
BUILD PROJECT Command	_BUILDER System Memory Variable	

C

_CALCVALUE System Memory Variable	CALL Command	CANCEL Command
CAPSLOCK( ) Function	CHANGE Command	CHRSAW( ) Function

_CLIPTEXT System Memory Variable
CLOSE MEMO Command
CNTBAR( ) Function
CNTPAD( ) Function
COL( ) Function
COMPILE Command
COMPILE DATABASE Command
COMPILE FORM Command
COMPOBJ( ) Function
Container Object
Control Object
_CONVERTER System Memory Variable
COPY FILE Command
CREATE CLASS Command
CREATE CLASSLIB Command
CREATE COLOR SET Command
CREATE Command
CREATE CONNECTION Command
CREATE DATABASE Command
CREATE FORM Command
CREATE FROM Command
CREATE LABEL Command
CREATE MENU Command
CREATE PROJECT Command
CREATE QUERY Command
CREATE REPORT Command
CREATE SCREEN Command
CREATE SQL VIEW Command
CREATE TRIGGER Command
CREATE VIEW Command
CREATEOBJECT( ) Function
_CUROBJ System Memory Variable

D

_DBLCLICK System Memory Variable
DBSETPROP( ) Function
DDE Functions
DEACTIVATE MENU Command
DEACTIVATE POPUP Command
DEACTIVATE WINDOW Command
DECLARE - DLL Command
DEFINE BAR Command
DEFINE BOX Command
DEFINE CLASS Command
DEFINE MENU Command
DEFINE PAD Command
DEFINE POPUP Command
DEFINE WINDOW Command
DELETE CONNECTION Command
DELETE DATABASE Command
DELETE TRIGGER Command
DELETE VIEW Command
_DIARYDATE System Memory Variable
DIR Command
DIRECTORY Command
DISPLAY Command
DISPLAY CONNECTIONS Command
DISPLAY DATABASE Command
DISPLAY DLLS Command
DISPLAY FILES Command
DISPLAY MEMORY Command
DISPLAY OBJECTS Command
DISPLAY PROCEDURES Command
DISPLAY STATUS Command
DISPLAY STRUCTURE Command
DISPLAY TABLES Command
DISPLAY VIEWS Command
DO FORM Command

E

EDIT Command
EJECT Command
EJECT PAGE Command
EXPORT Command
EXTERNAL Command

F

FILER Command
FIND Command
FKLABEL( ) Function
FKMAX( ) Function
FONTMETRIC( ) Function
_FOXDOC System Memory Variable
_FOXGRAPH System Memory Variable

G

_GENGRAPH System Memory Variable
_GENMENU System Memory Variable
_GENPD System Memory Variable
_GENSCRN System Memory Variable
_GENXTAB System Memory Variable
GETBAR( ) Function
GETCOLOR( ) Function
GETDIR( ) Function
GETFILE( ) Function
GETFONT( ) Function
GETOBJECT( ) Function
GETPAD( ) Function
GETPICT( ) Function
GETPRINTER( ) Function

H

HELP Command
HIDE MENU Command
HIDE POPUP Command
HIDE WINDOW Command

I

IMESTATUS( ) Function
IMPORT Command
_INDENT System Memory Variable
INDEX ON Command
INKEY( ) Function
INPUT Command
INSERT Command
INSMODE( ) Function
ISCOLOR( ) Function
ISMOUSE( ) Function

J

JOIN Command

K

KEYBOARD Command

L

LABEL Command
LASTKEY( ) Function
LIST Commands

M

N

O

P

Q

R

RGBSCHEME( ) Function
_RMARGIN System Memory Variable
ROW( ) Function
RUN | ! Command
RUNSCRIPT Command

S

SAVE MACROS Command
SAVE SCREEN Command
SAVE TO Command
SAVE WINDOWS Command
SCHEME( ) Function
SCOLS( ) Function
SCROLL Command
_SCREEN System Memory Variable
SET Command
SET ALTERNATE Command
SET ANSI Command
SET APLABOUT Command
SET AUTOSAVE Command
SET BELL Command
SET BLINK Command
SET BORDER Command
SET BRSTATUS Command
SET CLASSLIB Command
SET CLEAR Command
SET CLOCK Command
SET COLOR OF Command
SET COLOR OF SCHEME Command
SET COLOR SET Command
SET COLOR TO Command
SET COMPATIBLE Command
SET CONFIRM Command
SET CONSOLE Command
SET CPCOMPILE
SET CPDIALOG
SET CURRENCY Command
SET CURSOR Command
SET DATASESSION Command
SET DEBUG Command
SET DECIMALS Command
SET DELIMITERS Command
SET DEVELOPMENT Command
SET DEVICE Command
SET DISPLAY Command
SET DOHISTORY Command
SET ECHO Command
SET ESCAPE Command
SET FORMAT Command
SET FUNCTION Command
SET HEADINGS Command
SET HELP Command
SET HELPFILTER Command
SET INTENSITY Command
SET KEY Command
SET KEYCOMP Command
SET LOGERRORS Command
SET MACDESKTOP Command
SET MACHELP Command
SET MACKEY Command
SET MARGIN Command
SET MARK OF Command
SET MARK TO Command
SET MEMOWIDTH Command
SET MESSAGE Command
SET MOUSE Command
SET ODOMETER Command
SET OLEOBJECT Command
SET PALETTE Command
SET PDSETUP Command
SET POINT Command
SET PRINTER Command
SET READBORDER Command
SET REFRESH
SET RESOURCE
SET SAFETY

X

Y

Z

Supported ODBC API

The Visual FoxPro ODBC Driver supports:

- All Core Level API
- All Level 1 API
- Most Level 2 API

For more detail on any ODBC API function, refer to the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

Each API topic provides a brief summary and any Visual FoxPro-specific details. Note that several of the functions behave differently depending on whether the data source is defined as a connection to a directory of free tables (.DBF files) or to a Visual FoxPro database (.DBC file). Certain operations are only supported for database connections.

Core Level API Support

Level 1 API Support

Level 2 API Support

Core Level API Support

All of the ODBC Core Level API are supported. A brief description of each plus any details as they pertain to Visual FoxPro are available below.

SQLAllocConnect

SQLAllocEnv

SQLAllocStmt

SQLBindCol

SQLCancel

SQLColAttributes

SQLConnect

SQLDescribeCol

SQLDisconnect

SQLError

SQLExecDirect

SQLExecute

SQLFetch

SQLFreeConnect

SQLFreeEnv

SQLFreeStmt

SQLGetCursorName

SQLNumResultCols

SQLPrepare

SQLRowCount

SQLSetCursorName

SQLTransact

Level 1 API Support

All of the ODBC Level 1 API are supported. A brief description of each plus any details as they pertain to Visual FoxPro are available below.

SQLBindParameter

SQLColumns

SQLDriverConnect

SQLGetConnectOption

SQLGetData

SQLGetFunctions

SQLGetInfo

SQLGetStmtOption

SQLGetTypeInfo

SQLParamData

SQLPutData

SQLSetConnectOption

SQLSetStmtOption

SQLSpecialColumns

SQLStatistics

SQLTables

Level 2 API Support

The following ODBC Level 2 APIs are fully or partially supported:

SQLDataSources

SQLDrivers

SQLExtendedFetch

SQLMoreResults

SQLNumParams

SQLParamOptions

SQLPrimaryKeys

SQLSetPos

SQLSetScrollOptions (partial support)

The following Level 2 APIs are not supported:

SQLBrowseConnect

SQLColumnPrivileges

SQLDescribeParam

SQLForeignKeys

SQLNativeSql

SQLProcedureColumns

SQLProcedures

SQLTablePrivileges

SQLAllocConnect

Support: Full ODBC API Conformance: Core Level

Allocates memory for a connection handle, *hdbc*, within the environment identified by *henv*. The driver manager processes this call and calls the driver's **SQLAllocConnect** whenever **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect** is called.

For more information on **SQLAllocConnect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLAllocEnv

Support: Full ODBC API Conformance: Core Level

Allocates memory for an environment handle, *henv*, and initializes the ODBC call level interface for use by an application.

For more information on **SQLAllocEnv**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLAllocStmt

Support: Full ODBC API Conformance: Core Level

Allocates memory for a statement handle and associates the statement handle with the connection specified by *hdbc*. The Driver Manager passes this call to the driver, which allocates the memory for the *hstmt* structure.

For more information on **SQLAllocStmt**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLBindCol

Support: Full ODBC API Conformance: Core Level

Assigns storage space for a result column and specifies the type of the result. When **SQLFetch** or **SQLExtendedFetch** is called, the driver places the data for all bound columns in the assigned locations. See **SQLGetTypeInfo** for the mapping between ODBC and Visual FoxPro data types.

For more information on **SQLBindCol**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLBindParameter

Support: Full ODBC API Conformance: Level One

Associates a buffer with a parameter marker in a SQL statement. The Visual FoxPro ODBC Driver supports input parameters as specified by the *fParamType* argument.

For more information on **SQLBindParameter**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLCancel

Support: Full ODBC API Conformance: Core Level

Cancels the processing on a statement handle, *hstmt*.

For more information on **SQLCancel**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLColAttributes

Support: Full ODBC API Conformance: Core Level

Returns descriptor information for a column in a result set; Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

Note **SQLColAttributes** cannot be used to return information about the bookmark column (column 0).

The Visual FoxPro ODBC Driver supports all *fDescType* values. The following table includes comments on the driver's implementation of selected values:

fDescType	**Comment**
SQL_COLUMN_AUTO_INCREMENT	Returns FALSE: Visual FoxPro has no counter fields.
SQL_COLUMN_CASE_SENSITIVE	Always returns TRUE if the column type is Character.
SQL_COLUMN_LABEL	Returns the column name, which is also returned by SQL_COLUMN_NAME.
SQL_COLUMN_MONEY	Returns TRUE if the column type is Currency (represented by a "Y" in the Visual FoxPro language).
SQL_COLUMN_OWNER_NAME	Always returns an empty string.
SQL_COLUMN_SEARCHABLE	Returns SQL_UNSEARCHABLE for columns of type General; these columns cannot be used in a WHERE clause. Returns SQL_SEARCHABLE for columns of type Character or Memo with NOCPTRANS not set; these columns can be used in a WHERE clause with any comparison operator. Returns SQL_ALL_EXCEPT_LIKE for all other column types; these columns can be used in a WHERE clause with all comparison operators except LIKE.

For more information on **SQLColAttributes**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLColumns

Support: Full ODBC API Conformance: Level One

Creates a result set for a table which is the column list for the specified table or tables.

For more information on **SQLColumns**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLConnect

Support: Full ODBC API Conformance: Core Level

Connects to a data source, which can be either a database or a directory of tables. The Visual FoxPro ODBC Driver ignores the *szUID*, *cbUID*, *szAuthStr*, and *cbAuthStr* arguments.

For more information on **SQLConnect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLDataSources

Support: Full ODBC API Conformance: Level Two

Lists data source names.

For more information on **SQLDataSources**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLDescribeCol

Support: Full ODBC API Conformance: Core Level

Returns the name, type, precision, scale and nullability of the given result column.

For more information on **SQLDescribeCol**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLDisconnect

Support: Full ODBC API Conformance: Core Level

Closes a connection.

For more information on **SQLDisconnect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLDriverConnect

Support: Full ODBC API Conformance: Level One

Connects to an existing data source, which can be either a database or a directory of free tables. The ODBC attribute keywords UID and PWD are ignored. The following table lists the additional supported attribute keywords.

ODBC Attribute Keyword	Attribute Value
DSN	
UID	Ignored by the Visual FoxPro ODBC Driver, but does not generate an error.
PWD	Ignored by the Visual FoxPro ODBC Driver, but does not generate an error.
Driver	The name and location of the Visual FoxPro ODBC Driver; implemented by the driver manager.

Visual FoxPro ODBC Driver Attribute Keyword	Attribute Value
BackgroundFetch	"Yes" or "No"
Collate	"Machine" or other collating sequence. For a list of supported collating sequences, see SET COLLATE.
Description	
Exclusive	"Yes" or "No"
SourceDB	A fully qualified path to a directory containing zero or more free tables, or the absolute path and filename for a database.
SourceType	"DBC" or "DBF"
Version	

If the data source name is not specified, the driver manager prompts the user for the information (depending on the setting of the *fDriverCompletion* argument) and then continues. If more information is required, the Visual FoxPro ODBC Driver displays the prompt dialog.

For more information on **SQLDriverConnect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLDrivers

Support: Full ODBC API Conformance: Level Two

Lists driver descriptions and driver attribute keywords.

For more information on **SQLDrivers**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLError

Support: Full ODBC API Conformance: Core Level

Returns error or status information about the last error. The driver maintains a stack or list of errors that can be returned for the *hstmt*, *hdbc*, and *henv* arguments, depending on how the call to **SQLError** is made. The error queue is flushed after each statement.

The following table describes the **SQLError** arguments and return values used by the driver.

SQLError Argument	Return Value Description
szSQLState	The value for the SQLSTATE represented by the error.
pfNativeError	A non-zero value indicates a Visual FoxPro ODBC Driver Native Error Message. A value of zero indicates the error has been detected by the driver and mapped to the appropriate ODBC Error Code.
szErrorMsg	The text for the native error or ODBC error.
pcbErrorMsg	The length of the message text plus the length of the identifiers.

For more information on driver error messages, see Error Messages Overview. For more information on **SQLError**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLExecDirect

Support: Full ODBC API Conformance: Core Level

Executes a new, preparable SQL statement. The Visual FoxPro ODBC Driver uses the current values of the parameter marker variables if any parameters exist in the statement.

If you want to create a batch command to submit more than one SQL statement at a time, use a semicolon (;) to separate each SQL statement in the batch.

If your table, view or field names contain spaces, contain the names in back quote marks. For example, if your database contains a table named `My Table` and the field `My Field`, enclose each element of the identifier as shown below:

```
SELECT `My Table`.`Field1`, `My Table`.`Field2` FROM `My Table`
```

For more information on **SQLExecDirect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLExecute

Support: Full ODBC API Conformance: Core Level

Executes a prepared SQL statement (a statement already prepared by **SQLPrepare**). The driver uses the current values of the parameter marker variables if any parameters exist in the statement.

For more information on **SQLExecute**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLExtendedFetch

Support: Full ODBC API Conformance: Level Two

Similar to **SQLFetch**, but returns multiple rows using an array for each column. The result set is forward-scrollable and can be made backward-scrollable if the cursor is defined to be static, not forward-only.

By default, the Visual FoxPro ODBC Driver does not return rows marked as deleted in a FoxPro table. Rows marked for deletion but not yet removed from a table are not included in the result set cursor. You can change this behavior by using the SET DELETED command.

For more information on **SQLExtendedFetch**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLFetch

Support: Full ODBC API Conformance: Core Level

Retrieves one row from a result set into the locations specified by the previous calls to **SQLBindCol**. Prepares the driver for a call to **SQLGetData** for the unbound columns.

For more information on **SQLFetch**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLFreeConnect

Support: Full ODBC API Conformance: Core Level

Releases a connection handle and frees all memory allocated for the handle.

For more information on **SQLFreeConnect**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLFreeEnv

Support: Full ODBC API Conformance: Core Level

Closes the Visual FoxPro ODBC Driver, and releases all memory associated with the driver.

For more information on **SQLFreeEnv**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLFreeStmt

Support: Full ODBC API Conformance: Core Level

Stops processing associated with a specific *hstmt*, closes any open cursors associated with the *hstmt*, discards pending results, and, optionally, frees all resources associated with the statement handle.

For more information on **SQLFreeStmt**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLGetConnectOption

Support: Partial ODBC API Conformance: Level One

Returns the current setting of a connection option. This function is partially supported: the driver supports all values for the *fOption* argument, but does not support some of *vParam* values for the *fOption* argument SQL_TXN_ISOLATION.

For a complete list of *fOption* arguments, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

The following table describes only those arguments with behavior specific to the Visual FoxPro ODBC Driver implementation of **SQLGetConnectOption**:

fOption	**Remarks**
SQL_AUTOCOMMIT	If you choose SQL_AUTOCOMMIT_OFF, your application must explicitly commit or roll back transactions with **SQLTransact**; the Visual FoxPro ODBC Driver does not automatically commit a transactable statement upon completion. The driver does begin a transaction if the statement is transactable.
SQL_CURRENT_QUALIFIER	Can be a fully qualified database (.DBC file) name or fully qualified path to a directory containing zero or more tables (.DBF files).
SQL_LOGINTIMEOUT	Returns "Driver Not Capable" error
SQL_CURSORS	Returns "Driver Not Capable" error
SQL_PACKET_SIZE	Returns "Driver Not Capable" error
SQL_TXN_ISOLATION	The driver allows only: SQL_TXN_READ_COMMITTED; The following *vParam*s are not supported: SQL_TXN_READ_UNCOMMITTED; SQL_TXN_REAPEATABLE_READ; SQL_TXN_SERIALIZABLE

For more information on **SQLGetConnectOption**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLGetCursorName

Support: Full ODBC API Conformance: Core Level

Returns the name of the cursor associated with the given *hstmt*. **SQLGetCursorName** is included in the Visual FoxPro ODBC Driver API because it is a part of Core Level API functionality; it cannot be used with other API functions because the driver does not support positioned updates.

For more information on **SQLGetCursorName**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLGetData

Support: Full ODBC API Conformance: Level One

Retrieves the value of a single field in the current record of the given result set.

For more information on **SQLGetData**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLGetFunctions

Support: Full ODBC API Conformance: Level One

Returns TRUE for all supported functions.

The Visual FoxPro ODBC Driver supports all ODBC API Core and Level 1 functions. The following table indicates whether the driver supports a specific Level 2 function:

fFunction	**Supported**
SQL_API_SQLBROWSECONNECT	No
SQL_API_SQLCOLUMNPRIVELEGES	No
SQL_API_SQLDATASOURCES	Yes
SQL_API_SQLDESCRIBEPARAM	No
SQL_API_SQLDRIVERS	Yes
SQL_API_SQLEXTENDEDFETCH	Yes
SQL_API_SQLFOREIGNKEYS	No
SQL_API_SQLMORERESULTS	Yes
SQL_API_SQLNATIVESQL	No
SQL_API_SQLNUMPARAMS	Yes
SQL_API_SQLPARAMOPTIONS	Yes
SQL_API_SQLPRIMARYKEYS	Yes
SQL_API_SQLPROCEDURECOLUMNS	No
SQL_API_SQLPROCEDURES	No
SQL_API_SQLSETPOS	Yes
SQL_API_SQLSETSCROLLOPTIONS	Yes
SQL_API_SQLTABLEPRIVILEGES	No

For more information on **SQLGetFunctions**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLGetInfo

Support: Full ODBC API Conformance: Level One

Returns general information about the Visual FoxPro ODBC Driver and data source associated with a connection handle, *hdbc*. The following list shows the value returned by the Visual FoxPro ODBC Driver for each *fInfoType* argument, and comments regarding the returned values.

For more information on **SQLGetInfo**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

A

SQL_ACCESSIBLE_PROCEDURES returns 'N'.

SQL_ACCESSIBLE_TABLES returns 'Y'.

SQL_ACTIVE_CONNECTIONS returns 0.

SQL_ACTIVE_STATEMENTS returns 0.

SQL_ALTER_TABLE returns
SQL_AT_ADD_COLUMN
SQL_AT_DROP_COLUMN .

B

SQL_BOOKMARK_PERSISTENCE returns SQL_BP_SCROLL.

C

SQL_COLUMN_ALIAS returns 'Y'.

SQL_CONCAT_NULL_BEHAVIOR returns SQL_CB_NULL.

SQL_CONVERT_BIGINT returns 0. The Visual FoxPro ODBC Driver does not support BigInt.

SQL_CONVERT_BINARY returns 0.

SQL_CONVERT_BIT returns 0.

SQL_CONVERT_CHAR returns 0.

SQL_CONVERT_DATE returns 0.

SQL_CONVERT_DECIMAL returns 0.

SQL_CONVERT_DOUBLE returns 0.

SQL_CONVERT_FLOAT returns 0.

SQL_CONVERT_INTEGER returns 0.

SQL_CONVERT_LONGVARBINARY returns 0.

SQL_CONVERT_LONGVARCHAR returns 0.

SQL_CONVERT_NUMERIC returns 0.

SQL_CONVERT_REAL returns 0.

SQL_CONVERT_SMALLINT returns 0.

SQL_CONVERT_TIME returns 0.

SQL_CONVERT_TIMESTAMP returns 0.

SQL_CONVERT_TINYINT returns 0.

SQL_CONVERT_VARBINARY returns 0.

SQL_CONVERT_VARCHAR returns 0.

SQL_CONVERT_FUNCTIONS returns 0.

SQL_CORRELATION_NAME returns SQL_CN_ANY.

SQL_CURSOR_COMMIT_BEHAVIOR returns SQL_CB_PRESERVE.

SQL_CURSOR_ROLLBACK_BEHAVIOR returns SQL_CB_PRESERVE.

D

SQL_DATA_SOURCE_NAME returns the value passed as DSN to **SQLConnect**, or **SQLDriverConnect**; returns an empty string if no DSN is specified.

SQL_DATA_SOURCE_READ_ONLY returns 'N'.

SQL_DATABASE_NAME returns a full UNC path to the current database if the data source is a database. If the data source connects to a directory of tables, the function returns the path to the directory.

SQL_DBMS_NAME returns "Visual FoxPro".

SQL_DBMS_VER returns "03.00.0000".

SQL_DEFAULT_TXN_ISOLATION returns SQL_TXN_READ_COMMITTED. Dirty reads are not possible, but nonrepeatable reads and phantoms are possible.

SQL_DRIVER_HDBC is implemented by the Driver Manager.

SQL_DRIVER_HENV is implemented by the Driver Manager.

SQL_DRIVER_HLIB is implemented by the Driver Manager.

SQL_DRIVER_HSTMT is implemented by the Driver Manager.

SQL_DRIVER_NAME returns "VFPODBC.DLL".

SQL_DRIVER_ODBC_VER returns "02.50" (SQL_SPEC_MAJOR, SQL_SPEC_MINOR).

SQL_DRIVER_VER returns "01.00.0000".

E

SQL_EXPRESSIONS_IN_ORDERBY returns 'N'.

F

SQL_FETCH_DIRECTION returns
SQL_FD_FETCH_NEXT
SQL_FD_FETCH_FIRST
SQL_FD_FETCH_LAST
SQL_FD_FETCH_PRIOR
SQL_FD_FETCH_ABSOLUTE
SQL_FD_FETCH_RELATIVE
SQL_FD_FETCH_BOOKMARK.

SQL_FILE_USAGE returns SQL_FILE_QUALIFIER for both database (.DBC file) and free table (.DBF file) data sources.

G-H

SQL_GETDATA_EXENSIONS returns
SQL_GD_ANY_COLUMN
SQL_GD_ANY_BLOCK
SQL_GD_ANY_BOUND
SQL_GD_ANY_ORDER.

SQL_GROUP_BY returns SQL_GB_NO_RELATION.

I-J

SQL_IDENTIFIER_CASE returns SQL_IC_MIXED.
SQL_IDENTIFIER_QUOTE_CHAR returns `.

K

SQL_KEYWORDS returns “”.

L

SQL_LIKE_ESCAPE_CLAUSE returns ‘N’.
SQL_LOCK_TYPES returns SQL_LCK_NO_CHANGE.

M

SQL_MAX_BINARY_LITERAL_LEN returns 0.
SQL_MAX_CHAR_LITERAL_LEN returns 254.
SQL_MAX_COLUMN_NAME_LEN returns 128.
SQL_MAX_COLUMNS_IN_GROUP_BY returns 16.
SQL_MAX_COLUMNS_IN_ORDER_BY returns 16.
SQL_MAX_COLUMNS_IN_INDEX returns 0.
SQL_MAX_COLUMNS_IN_SELECT returns 254.
SQL_MAX_COLUMNS_IN_TABLE returns 254.
SQL_MAX_CURSOR_NAME_LEN returns 254.
SQL_MAX_INDEX_SIZE returns 0.
SQL_MAX_OWNER_NAME_LEN returns 0.
SQL_MAX_PROCEDURE_NAME_LEN returns 0. The Visual FoxPro ODBC Driver does not allow direct access to Visual FoxPro stored procedures.
SQL_MAX_QUALIFIER_NAME_LEN returns the maximum operating system path length.
SQL_MAX_ROW_SIZE returns 254^2.
SQL_MAX_ROW_SIZE_INCLUDES_LONG returns ‘N’.
SQL_MAX_STATEMENT_LEN returns 8192.
SQL_MAX_TABLE_NAME_LEN returns 128.
SQL_MAX_TABLES_IN_SELECT returns 16.
SQL_MAX_USER_NAME_LEN returns 0.
SQL_MULT_RESULT_SETS returns ‘Y’.
SQL_MULTIPLE_ACTIVE_TXN returns ‘Y’. Multiple connections can have transactions open at once.

N

SQL_NEED_LONG_DATA_LEN returns ‘N’.
SQL_NON_NULLABLE_COLUMNS returns SQL_NNC_NON_NULL.
SQL_NULL_COLLATION returns SQL_NC_LOW.
SQL_NUMERIC_FUNCTIONS returns all functions except SQL_FN_NUM_POWER, which is not supported by Visual FoxPro ODBC Driver. The following functions are supported:
SQL_FN_NUM_ABS
SQL_FN_NUM_ACOS
SQL_FN_NUM_ASIN
SQL_FN_NUM_ATAN
SQL_FN_NUM_ATAN2
SQL_FN_NUM_CELING
SQL_FN_NUM_COS

SQL_FN_NUM_COT
SQL_FN_NUM_DEGREES
SQL_FN_NUM_EXP
SQL_FN_NUM_FLOOR
SQL_FN_NUM_LOG
SQL_FN_NUM_LOG10
SQL_FN_NUM_MOD
SQL_FN_NUM_PI
SQL_FN_NUM_RADIANS
SQL_FN_NUM_RAND
SQL_FN_NUM_ROUND
SQL_FN_NUM_SIGN
SQL_FN_NUM_SIN
SQL_FN_NUM_SQRT
SQL_FN_NUM_TAN.

O

SQL_ODBC_API_CONFORMANCE returns SQL_OAC_LEVEL1.

SQL_ODBC_SAG_CLI_CONFORMANCE returns SQL_OSCC_COMPLIANT. .

SQL_ODBC_SQL_CONFORMANCE returns SQL_OSC_MINIMUM. Minimum SQL syntax is supported .

SQL_ODBC_SQL_OPT_IEF returns “N”. .

SQL_ODBC_VER is implemented by the Driver Manager.

SQL_ORDER_BY_COLUMNS_IN_SELECT returns “N”.

SQL_OUTER_JOINS returns “N”.

SQL_OWNER_TERM returns “”. The Visual FoxPro ODBC Driver does not support owners for its objects.

SQL_OWNER_USAGE returns 0. The Visual FoxPro ODBC Driver does not support owners for its objects.

P

SQL_POS_OPERATIONS returns SQL_POS_POSITION.

SQL_POSITIONED_STATEMENTS returns 0.

SQL_PROCEDURE_TERM returns “”.

SQL_PROCEDURES returns ‘N’.

Q

SQL_QUALIFIER_LOCATION returns SQL_QL_START.

SQL_QUALIFIER_NAME_SEPARATOR returns ‘!’ or ‘\’. The separator between database and table is ‘!’ for data sources connected to databases, and ‘\’ for data sources that are directories of free tables.

SQL_QUALIFIER_TERM returns “database” or “directory”. The qualifier is “database” for data sources connected to databases, and “directory” for data sources that are directories of free tables.

SQL_QUALIFIER_USAGE does not support SQL_QU_PRIVILEGE_DEFINITION; it returns:
SQL_QU_DML_STATEMENT
SQL_QU_TABLE_DEFINITION .

SQL_QUOTED_IDENTIFIER_CASE returns SQL_IC_MIXED.

R

SQL_ROW_UPDATES returns “N”. The Visual FoxPro ODBC Driver supports only static and forward cursors.

S

SQL_SCROLL_CONCURRENCY returns SQL_SCCO_READ_ONLY.

SQL_SCROLL_OPTIONS returns
SQL_SO_STATIC
SQL_SO_READONLY.

SQL_SEARCH_PATTERN_ESCAPE returns “\”.

SQL_SERVER_NAME returns “”.

SQL_SPECIAL_CHARACTERS returns “~@#$%^”.

SQL_STATIC_SENSITIVITY returns 0. The Visual FoxPro ODBC Driver does not supports positional updates.

SQL_STRING_FUNCTIONS does not support SQL_FN_STR_INSERT, SQL_FN_STR_LOCATE, SQL_FN_STR_LOCATE_2, or SQL_FN_STR_SOUNDEX.
It returns
SQL_FN_STR_ASCII
SQL_FN_STR_CHAR
SQL_FN_STR_CONCAT
SQL_FN_STR_DIFFERENCE
SQL_FN_STR_LCASE
SQL_FN_STR_LEFT
SQL_FN_STR_LENGTH
SQL_FN_STR_LTRIM
SQL_FN_STR_REPEAT
SQL_FN_STR_REPLACE
SQL_FN_STR_RIGHT
SQL_FN_STR_RTRIM
SQL_FN_STR_SUBSTRING
SQL_FN_STR_UCASE
SQL_FN_STR_SPACE.

SQL_SUBQUERIES returns
SQL_SQ_CORRELATED_SUBQUERIES
SQL_SQ_COMPARISON
SQL_SQ_EXISTS
SQL_SQ_IN
SQL_SQ_QUANTIFIED.

SQL_SYSTEM_FUNCTIONS returns
SQL_FN_SYS_DBNAME
SQL_FN_SYS_IFNULL
but not
SQL_FN_SYS_USERNAME.

T

SQL_TABLE_TERM returns “table”.

SQL_TIMEDATE_ADD_INTERVALS returns
SQL_FN_TSI_ SECOND
SQL_FN_TSI_MINUTE
SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY
SQL_FN_TSI_MONTH
SQL_FN_TSI_YEAR

Not SQL_FN_TSI_FRAC_SECOND, SQL_FN_TSI_WEEK, or SQL_FN_TSI_QUARTER.

SQL_TIMEDATE_DIFF_INTERVALS returns
SQL_FN_TSI_ SECOND
SQL_FN_TSI_MINUTE
SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY
SQL_FN_TSI_MONTH
SQL_FN_TSI_YEAR.

SQL_TIMEDATE_FUNCTIONS does not support SQL_FN_TD_QUARTER, SQL_FN_TD_TIMESTAMPADD, SQL_FN_TD_DAYOFYEAR, or SQL_FN_TD_WEEK. It returns
SQL_FN_TD_CURDATE
SQL_FN_TD_CURTIME
SQL_FN_TD_DAYNAME
SQL_FN_TD_DAYOFMONTH
SQL_FN_TD_DAYOFWEEK
SQL_FN_TD_HOUR
SQL_FN_TD_MINUTE
SQL_FN_TD_MONTH
SQL_FN_TD_MONTHNAME
SQL_FN_TD_NOW
SQL_FN_TD_SECOND
SQL_FN_TD_TIMESTAMPDIFF
SQL_FN_TD_YEAR .

SQL_TXN_CAPABLE returns SQL_TC_DML.

SQL_TXN_ISOLATION_OPTION returns SQL_TXN_READ_COMMITTED.

U-Z

SQL_UNION returns
SQL_U_UNION
SQL_U_UNION_ALL.

SQL_USER_NAME returns <blank>.

SQLGetStmtOption

Support: Full ODBC API Conformance: Level One

Returns the current setting of a statement option.

fOption	**Returns**
SQL_GET_BOOKMARK	32-bit integer value that is the bookmark for the current record number
SQL_ROW_NUMBER	32 bit integer specifying the position of the current row within the result set
SQL_TRANSLATE_DLL	Error: “Driver not capable”.

The Visual FoxPro ODBC Driver has no translation DLLs.

For more information on **SQLGetStmtOption**, see the *Microsoft ODBC 3.0 Programmer’s Reference and SDK Guide*.

SQLGetTypeInfo

Support: Full ODBC API Conformance: Level One

Returns information about the data types supported by a data source. The driver returns the information in a SQL result set. The following table lists ODBC data types and the corresponding Visual FoxPro data type.

ODBC Type	Visual FoxPro Type
SQL_BIGINT	Not supported. There is no 64-bit Visual FoxPro type.
SQL_BIT	Logical
SQL_CHAR	Character
SQL_DATE	Date
SQL_DECIMAL	Numeric
SQL_DOUBLE	Double
SQL_FLOAT	Double
SQL_INTEGER	Integer
SQL_LONGVARBINARY	Memo (Binary)
SQL_LONGVARCHAR	Memo
SQL_NUMERIC	Numeric*, Currency, Float
SQL_REAL	Double
SQL_SMALLINT	Integer
SQL_TIME	Not supported. There is no Visual FoxPro *time* type.
SQL_TIMESTAMP	DateTime
SQL_TINYINT	Integer
SQL_VARBINARY	Memo (Binary)*, eneral
SQL_VARCHAR	Character

*Default type

For more information on Visual FoxPro data types, see CREATE TABLE. For more information on **SQLGetTypeInfo**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLMoreResults

Support: Full ODBC API Conformance: Level Two

Determines whether more results are pending on a statement handle, *hstmt*, containing SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.

For more information on **SQLMoreResults**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLNumParams

Support: Full ODBC API Conformance: Level Two

Returns the number of parameters in a SQL statement. The number of parameters should equal the number of question marks in the SQL statement passed to **SQLPrepare**.

For more information on SQL grammar, see Supported ODBC SQL Grammar. For more information on **SQLNumParams**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLNumResultCols

Support: Full ODBC API Conformance: Core Level

Returns the number of columns in a result set cursor.

For more information on **SQLNumResultCols**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLParamData

Support: Full ODBC API Conformance: Level One

Used in conjunction with **SQLPutData** to specify parameter data at statement execution time.

For more information on **SQLParamData**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLParamOptions

Support: Full ODBC API Conformance: Level One

Allows an application to specify multiple values for the set of parameters assigned by **SQLBindParameter**. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. An application can, for example, specify three sets of values for the set of parameters associated with an INSERT statement, and then execute the INSERT statement once to perform the three insert operations.

For more information on **SQLParamOptions**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLPrepare

Support: Full ODBC API Conformance: Core Level

Prepares a SQL statement by planning how to optimize and execute the statement. The SQL statement is compiled for execution by **SQLExecDirect**.

If your table, view or field names contain spaces, contain the names in back quote (`) marks. For example, if your database contains a table named `My Table` and the field `My Field`, enclose each element of the identifier as shown below:

```
SELECT * FROM `My Table`.`My Field`
```

For more information on **SQLPrepare**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLPrimaryKeys

Support: Full ODBC API Conformance: Level Two

Returns the column names that comprise the primary key for a table. The Visual FoxPro ODBC Driver implementation of **SQLPrimaryKeys**:

- Ignores the *szTableOwner* and *cbTableOwner* arguments.
- Only works for data sources that are databases. The driver returns the error “Driver does not support this function” if the data source is a directory of free tables.

For more information on **SQLPrimaryKeys**, see the *Microsoft ODBC 3.0 Programmer’s Reference and SDK Guide*.

SQLPutData

Support: Full ODBC API Conformance: Level One

Allows an application to send data for a parameter or column to the driver at statement execution time.

For more information on **SQLPutData**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLRowCount

Support: Full ODBC API Conformance: Core Level

Returns the number of rows affected by the last UPDATE, INSERT, or DELETE statement.

For more information on **SQLRowCount**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSetConnectOption

Support: Partial ODBC API Conformance: Level One

Sets options that govern aspects of connections. This function is partially supported: the driver supports all values for the *fOption* argument, but does not support some of *vParam* values for the *fOption* argument SQL_TXN_ISOLATION.

For a complete list of *fOption* arguments, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

The following table describes only those arguments with behavior specific to the Visual FoxPro ODBC Driver implementation of **SQLSetConnectOption**:

fOption	**Remarks**
SQL_AUTOCOMMIT	If you choose SQL_AUTOCOMMIT_OFF, your application must explicitly commit or roll back transactions with **SQLTransact**; the Visual FoxPro ODBC Driver does not automatically commit a transactable statement upon completion. The driver does begin a transaction if the statement is transactable.
SQL_CURRENT_QUALIFIER	Can be a fully qualified databasename or fully qualified path to a directory containing zero or more free tables.
SQL_LOGINTIMEOUT	Returns "Driver not capable" error
SQL_CURSORS	Returns "Driver not capable" error
SQL_PACKET_SIZE	Returns "Driver not capable" error
SQL_TXN_ISOLATION	The driver allows only: SQL_TXN_READ_COMMITTED; The following *vParam*s are not supported: SQL_TXN_READ_UNCOMMITTED; SQL_TXN_REAPEATABLE_READ; SQL_TXN_SERIALIZABLE

For more information on **SQLSetConnectOption**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSetCursorName

Support: Full ODBC API Conformance: Core Level

Associates a cursor name with an active statement handle, *hstmt*. **SQLSetCursorName** is included in the Visual FoxPro ODBC Driver API because it is a part of Core Level ODBC API functionality; it cannot be used with other API functions because the driver does not support positioned updates.

For more information on **SQLSetCursorName**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSetPos

Support: Full ODBC API Conformance: Level Two

Sets the cursor position in a rowset. You can use **SQLSetPos** with **SQLGetData** to retrieve rows from unbound columns after positioning the cursor to a specific row in the rowset.

For more information on **SQLSetPos**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSetScrollOptions

Support: Partial ODBC API Conformance: Level Two

Sets options that control the behavior of cursors associated with a statement handle, *hstmt*.

The Visual FoxPro ODBC Driver supports only SQL_CONCUR_READ_ONLY; it does not support the *fConcurrency* value SQL_CONCUR_ROWVER. The driver converts SQL_KEYSET_SIZE, SQL_CURSOR_DYNAMIC and SQL_CURSOR_KEYSET_DRIVEN to SQL_SCROLL_STATIC with warning ODBC_01S02.

For more information on **SQLSetScrollOptions**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSetStmtOption

Support: Full ODBC API Conformance: Level One

Sets options related to a statement handle, *hstmt*.

fOption	**Allowed Values**	**Comments**
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF	If you attempt to set this *fOption*, the driver returns the error: "Driver not capable". Visual FoxPro does not support asynchronous execution.
SQL_BIND_TYPE	SQL_BIND_BY_COLUMN or a 32-bit value denoting the length of the structure or an instance of a buffer into which result columns will be bound.	
SQL_CONCURRENCY	SQL_CONCUR_READ_ONLY SQL_CONCUR_LOCK SQL_CONCUR_VALUES	The driver doesn't allow SQL_CONCUR_ROWVER, since Visual FoxPro does not have row versioning based on timestamps.
SQL_CURSOR_TYPE	SQL_CURSOR_FORWARD_ONLY SQL_CURSOR_STATIC	The driver does not allow SQL_CURSOR_KEYSET_DRIVEN or SQL_CURSOR_DYNAMIC; see **SQLSetScrollOptions** for more information.
SQL_KEYSET_SIZE	Error: "Driver not capable"	Visual FoxPro does not support the keyset cursor model.
SQL_MAX_LENGTH	0	If you attempt to set this *fOption* value, the driver returns the error "Driver not capable".
SQL_MAX_ROWS	0	If you attempt to set this *fOption* value, the driver returns the error "Driver not capable".
SQL_NOSCAN	SQL_NOSCAN_OFF	
SQL_QUERY_TIMEOUT	0	If you attempt to set this *fOption* value, the driver returns the error "Driver not capable".
SQL_RETRIEVE_DATA	SQL_RD_ON, SQL_RD_OFF	
SQL_ROWSET_SIZE	1 to 4,294,967,296	
SQL_SIMULATE_CURSOR	Error: "Driver not capable"	
SQL_USE_BOOKMARKS	SQL_UB_OFF SQL_UB_ON	

For more information on **SQLSetStmtOption**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLSpecialColumns

Support: Full ODBC API Conformance: Level One

Retrieves the optimal set of columns that uniquely identifies a row in the table.

The Visual FoxPro ODBC Driver returns the columns that make up the primary key on the FoxPro table (see **SQLPrimaryKeys**). If called with *fColType* set to SQL_ROWVER, no columns are returned. **SQLSpecialColumns** only works for data sources that are databases.

For more information on **SQLSpecialColumns**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLStatistics

Support: Full ODBC API Conformance: Level One

Retrieves a list of statistics about a single table and the indexes, or tag names, associated with the table. The driver returns the information as a result set.

For more information on **SQLStatistics**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLTables

Support: Full ODBC API Conformance: Level One

Returns the list of table names specified by the parameter in the **SQLTables** statement. If no parameter is specified, returns the table names stored in the current data source. The driver returns the information as a result set.

Enumeration type calls will not receive a result set entry for remote views or local parameterized views. However, a call to **SQLTables** with a unique table name specifier will find a match for such a view if present with that name; this allows the API to used to check for name conflicts prior to creation of a new table.

Note The Visual FoxPro ODBC driver differentiates between database tables and free tables, even when both types of tables are stored in the same directory on your system. If your data source is a directory of free tables, the Visual FoxPro ODBC Driver does not catalog or return the names of any tables that are associated with a database.

For more information on **SQLTables**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

SQLTransact

Support: Full ODBC API Conformance: Core Level

Requests a commit or rollback operation for all active operations on all statement handles (*hstmt*s) associated with a connection, or all connections associated with the environment handle, *henv*. **SQLTransact** only works for data sources that are databases.

If a commit fails when in manual mode, the transaction remains active; you can choose to rollback the transaction or retry the commit operation. If a commit operation fails when in automatic transaction mode, the transaction is rolled back automatically; the transaction cannot be inactive.

For more information on **SQLTransact**, see the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

database

In Visual FoxPro, a database file has a .DBC extension and can contain one or more tables.

database table

In Visual FoxPro, a table that is associated with a database. Contrast free table.

free table

In Visual FoxPro, a table that is not associated with a database.

A .DBF file created in FoxPro version 2.x is a free table, unless it is converted to a Visual FoxPro table and added to a Visual FoxPro database. Contrast database table.

preparable SQL statement

A SQL statement that has not already been processed by **SQLPrepare**.

table

In Visual FoxPro, records are stored in a table. Each row of a table represents a record and the columns of the table represent the fields of the record. Each Visual FoxPro table is stored in its own file with a .DBF extension. Visual FoxPro tables can be associated with a database.

FoxPro versions 2.*x* tables are not associated with a database.

ALTER TABLE - SQL Command

See Also

Programmatically modifies the structure of a table.

Syntax

```
ALTER TABLE TableName1
   ADD | ALTER [COLUMN] FieldName1
      FieldType [(nFieldWidth [, nPrecision])]
      [NULL | NOT NULL]
      [CHECK lExpression1 [ERROR cMessageText1]]
      [DEFAULT eExpression1]
      [PRIMARY KEY | UNIQUE]
      [REFERENCES TableName2 [TAG TagName1]]
      [NOCPTRANS]
 - Or -
ALTER TABLE TableName1
   ALTER [COLUMN] FieldName2
      [NULL | NOT NULL]
      [SET DEFAULT eExpression2]
      [SET CHECK lExpression2 [ERROR cMessageText2]]
      [DROP DEFAULT]
      [DROP CHECK]
 - Or -
ALTER TABLE TableName1
   [DROP [COLUMN] FieldName3]
   [SET CHECK lExpression3 [ERROR cMessageText3]]
   [DROP CHECK]
   [ADD PRIMARY KEY eExpression3 TAG TagName2]
   [DROP PRIMARY KEY]
   [ADD UNIQUE eExpression4 [TAG TagName3]]
   [DROP UNIQUE TAG TagName4]
   [ADD FOREIGN KEY [eExpression5] TAG TagName4
      REFERENCES TableName2 [TAG TagName5]]
   [DROP FOREIGN KEY TAG TagName6 [SAVE]]
   [RENAME COLUMN FieldName4 TO FieldName5]
   [NOVALIDATE]
```

Arguments

TableName1 Specifies the name of the table whose structure is modified.

ADD [COLUMN] *FieldName1* Specifies the name of the field to add.

ALTER [COLUMN] *FieldName1* Specifies the name of an existing field to modify.

FieldType [(*nFieldWidth* [, *nPrecision*]]) Specifies the field type, field width, and field precision (number of decimal places) for a new or modified field.

FieldType is a single letter indicating the field's data type. Some field data types require that you specify *nFieldWidth* or *nPrecision* or both.

nFieldWidth and *nPrecision* are ignored for D, T, I, Y, L, M, G, and P types. *nPrecision* defaults to zero (no decimal places) if *nPrecision* isn't included for the N, F, or B types.

NULL | NOT NULL Allows or prevents null values in the field.

If you omit NULL and NOT NULL, the current setting of SET NULL determines if null values are allowed in the field. However, if you omit NULL and NOT NULL and include the PRIMARY KEY or

UNIQUE clause, the current setting of SET NULL is ignored and the field defaults to NOT NULL.

CHECK *lExpression1* Specifies a validation rule for the field. *lExpression1* must evaluate to a logical expression, and can be a user-defined function or a stored procedure. Note that when a blank record is appended, the validation rule is checked. An error is generated if the validation rule doesn't allow for a blank field value in an appended record.

ERROR *cMessageText1* Specifies the error message displayed when the field validation rule generates an error.

DEFAULT *eExpression1* Specifies a default value for the field. The data type of *eExpression1* must be the same as the data type for the field.

PRIMARY KEY Creates a primary index tag. The index tag has the same name as the field.

UNIQUE Creates a candidate index tag with the same name as the field.

Note Candidate indexes (created by including the UNIQUE option, provided for ANSI compatibility in ALTER TABLE or CREATE TABLE) are not the same as indexes created with the UNIQUE option in the INDEX command. An index created with UNIQUE in the INDEX command allows duplicate index keys; candidate indexes do not allow duplicate index keys.

Null values and duplicate records are not permitted in a field used for a primary or candidate index.

If you are creating a new field with ADD COLUMN, Visual FoxPro will not generate an error if you create a primary or candidate index for a field that supports null values. However, Visual FoxPro will generate an error if you attempt to enter a null or duplicate value into a field used for a primary or candidate index.

If you are modifying an existing field and the primary or candidate index expression consists of fields in the table, Visual FoxPro checks the fields to see if they contain null values or duplicate records. If they do, Visual FoxPro generates an error and the table is not altered.

REFERENCES *TableName2* TAG *TagName1* Specifies the parent table to which a persistent relationship is established. TAG *TagName1* specifies the parent table's index tag on which the relationship is based. Index tag names can contain up to 10 characters.

NOCPTRANS Prevents translation to a different code page for character and memo fields. If the table is converted to another code page, the fields for which NOCPTRANS has been specified are not translated. NOCPTRANS can only be specified for character and memo fields.

The following example creates a table named MYTABLE containing two character fields and two memo fields. The second character field `char2` and the second memo field `memo2` include NOCPTRANS to prevent translation.

```
CREATE TABLE mytable (char1 C(10), char2 C(10) NOCPTRANS,;
   memo1 M, memo2 M NOCPTRANS)
```

ALTER [COLUMN] *FieldName2* Specifies the name of an existing field to modify.

SET DEFAULT *eExpression2* Specifies a new default value for an existing field. The data type of *eExpression2* must be the same as the data type for the field.

SET CHECK *lExpression2* Specifies a new validation rule for an existing field. *lExpression2* must evaluate to a logical expression, and may be a user-defined function or a stored procedure.

ERROR *cMessageText2* Specifies the error message displayed when the field validation rule generates an error. The message is displayed only when data is changed within a Browse or Edit window.

DROP DEFAULT Removes the default value for an existing field.

DROP CHECK Removes the validation rule for an existing field.

DROP [COLUMN] *FieldName3* Specifies a field to remove from the table. Removing a field from the table also removes the field's default value setting and field validation rule.

If index key or trigger expressions reference the field, the expressions become invalid when the field is removed. In this case, an error isn't generated when the field is removed, but the invalid index key or trigger expressions will generate errors at run time.

SET CHECK *lExpression3* Specifies the table validation rule. *lExpression3* must evaluate to a

logical expression, and may be a user-defined function or a stored procedure.

ERROR *cMessageText3* Specifies the error message displayed when the table validation rule generates an error. The message is only displayed when data is changed within a Browse or Edit window.

DROP CHECK Removes the table's validation rule.

ADD PRIMARY KEY *eExpression3* TAG *TagName2* Adds a primary index to the table. *eExpression3* specifies the primary index key expression, and *TagName2* specifies the name of the primary index tag. Index tag names can contain up to 10 characters. If TAG *TagName2* is omitted and *eExpression3* is a single field, the primary index tag has the same name as the field specified in *eExpression3*.

DROP PRIMARY KEY Removes the primary index and its index tag. Because a table can have only one primary key, it isn't necessary to specify the name of the primary key. Removing the primary index also deletes any persistent relations based on the primary key.

ADD UNIQUE *eExpression4* [TAG *TagName3*] Adds a candidate index to the table. *eExpression4* specifies the candidate index key expression, and *TagName3* specifies the name of the candidate index tag. Index tag names can contain up to 10 characters. If you omit TAG *TagName3,* and if *eExpression4* is a single field, the candidate index tag has the same name as the field specified in *eExpression4*.

DROP UNIQUE TAG *TagName4* Removes the candidate index and its index tag. Because a table can have multiple candidate keys, you must specify the name of the candidate index tag.

ADD FOREIGN KEY [*eExpression5*] TAG *TagName4* Adds a foreign (non-primary) index to the table. *eExpression5* specifies the foreign index key expression and *TagName4* specifies the name of the foreign index tag. Index tag names can contain up to 10 characters.

REFERENCES *TableName2* [TAG *TagName5*] Specifies the parent table to which a persistent relationship is established. Include TAG *TagName5* to establish a relation based on an existing index tag for the parent table. Index tag names can contain up to 10 characters. If you omit TAG *TagName5*, the relationship is established using the parent table's primary index tag.

DROP FOREIGN KEY TAG *TagName6* [SAVE] Deletes a foreign key whose index tag is *TagName6*. If you omit SAVE, the index tag is deleted from the structural index. Include SAVE to prevent the index tag from being deleted from the structural index.

RENAME COLUMN *FieldName4* TO *FieldName5* Allows you to change the name of a field in the table. *FieldName4* specifies the name of the field that is renamed. *FieldName5* specifies the new name of the field.

Caution Exercise care when renaming table fields – index expressions, field and table validation rules, commands, functions, and so on may reference the original field names.

NOVALIDATE Specifies that Visual FoxPro allows changes to be made to the structure of the table that may violate the integrity of the data in the table. By default, Visual FoxPro prevents ALTER TABLE from making changes to the structure of the table that violates the integrity of the data in the table. Include NOVALIDATE to override this default behavior.

Remarks

ALTER TABLE can be used to modify the structure of a table that has not been added to a database. However, Visual FoxPro generates an error if you include the DEFAULT, FOREIGN KEY, PRIMARY KEY, REFERENCES, or SET clauses when modifying a free table.

ALTER TABLE may rebuild the table by creating a new table header and appending records to the table header. For example, changing a field's type or width may cause the table to be rebuilt.

After a table is rebuilt, field validation rules are executed for any fields whose type or width are changed. If you change the type or width of any field in the table, the table rule is executed.

If you modify field or table validation rules for a table that has records, Visual FoxPro tests the new field or table validation rules against the existing data and issues a warning on the first occurrence of

a field or table validation rule or a trigger violation.

If the table you modify is in a database, ALTER TABLE - SQL requires exclusive use of the database. To open a database for exclusive use, include EXCLUSIVE in OPEN DATABASE.

See Also

CREATE TABLE - SQL

INDEX

CREATE TABLE - SQL Command

See Also

Creates a table having the specified fields.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

```
CREATE TABLE | DBF TableName1 [NAME LongTableName] [FREE]
   (FieldName1 FieldType [(nFieldWidth [, nPrecision])]
      [NULL | NOT NULL]
      [CHECK lExpression1 [ERROR cMessageText1]]
      [DEFAULT eExpression1]
      [PRIMARY KEY | UNIQUE]
      [REFERENCES TableName2 [TAG TagName1]]
      [NOCPTRANS]
   [, FieldName2 ...]
      [, PRIMARY KEY eExpression2 TAG TagName2
      |, UNIQUE eExpression3 TAG TagName3]
      [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
         REFERENCES TableName3 [TAG TagName5]]
      [, CHECK lExpression2 [ERROR cMessageText2]])
| FROM ARRAY ArrayName
```

Arguments

CREATE TABLE | DBF *TableName1* Specifies the name of the table to create. The TABLE and DBF options are identical.

NAME *LongTableName* Specifies a long name for the table. A long table name can be specified only when a database is open because long table names are stored in databases.

Long names can contain up to 128 characters and can be used in place of short file names in the database.

FREE Specifies that the table will not be added to an open database. FREE isn't required if a database isn't open.

(FieldName1 FieldType [(*nFieldWidth* [, *nPrecision*])] Specifies the field name, field type, field width, and field precision (number of decimal places), respectively.

FieldType is a single letter indicating the field's data type. Some field data types require that you specify *nFieldWidth* or *nPrecision,* or both.

nFieldWidth and *nPrecision* are ignored for D, T, I, Y, L, M, G, and P types. *nPrecision* defaults to zero (no decimal places) if *nPrecision* isn't included for the N, F, or B types.

NULL Allows null values in the field.

NOT NULL Prevents null values in the field.

If you omit NULL and NOT NULL, the current setting of SET NULL determines if null values are allowed in the field. However, if you omit NULL and NOT NULL and include the PRIMARY KEY or UNIQUE clause, the current setting of SET NULL is ignored and the field defaults to NOT NULL.

CHECK *lExpression1* Specifies a validation rule for the field. *lExpression1* can be a user-defined function. Note that when a blank record is appended, the validation rule is checked. An error is generated if the validation rule doesn't allow for a blank field value in an appended record.

ERROR *cMessageText1* Specifies the error message Visual FoxPro displays when the field rule generates an error. The message is only displayed when data is changed within a Browse window or Edit window.

DEFAULT *eExpression1* Specifies a default value for the field. The data type of *eExpression1*must be the same as the field's data type.

PRIMARY KEY Creates a primary index for the field. The primary index tag has the same name as the field.

UNIQUE Creates a candidate index for the field. The candidate index tag has the same name as the field.

Note Candidate indexes (created by including the UNIQUE option in CREATE TABLE or ALTER TABLE - SQL) are not the same as indexes created with the UNIQUE option in the INDEX command. An index created with the UNIQUE option in the INDEX command allows duplicate index keys; candidate indexes do not allow duplicate index keys. See INDEX for additional information on its UNIQUE option.

Null values and duplicate records are not permitted in a field used for a primary or candidate index. However, Visual FoxPro will not generate an error if you create a primary or candidate index for a field that supports null values. Visual FoxPro will generate an error if you attempt to enter a null or duplicate value into a field used for a primary or candidate index.

REFERENCES *TableName2* [TAG *TagName1*] Specifies the parent table to which a persistent relationship is established. If you omit TAG *TagName1*, the relationship is established using the primary index key of the parent table. If the parent table does not have a primary index, Visual FoxPro generates an error.

Include TAG *TagName1* to establish a relation based on an existing index tag for the parent table. Index tag names can contain up to 10 characters.

The parent table cannot be a free table.

NOCPTRANS Prevents translation to a different code page for character and memo fields. If the table is converted to another code page, the fields for which NOCPTRANS has been specified are not translated. NOCPTRANS can only be specified for character and memo fields.

The following example creates a table named MYTABLE containing two character fields and two memo fields. The second character field CHAR2 and the second memo field MEMO2 include NOCPTRANS to prevent translation.

```
CREATE TABLE mytable (char1 C(10), char2 C(10) NOCPTRANS,;
   memo1 M, memo2 M NOCPTRANS)
```

PRIMARY KEY *eExpression2* TAG *TagName2* Specifies a primary index to create. *eExpression2* specifies any field or combination of fields in the table. TAG *TagName2 s*pecifies the name for the primary index tag that is created. Index tag names can contain up to 10 characters.

Because a table can have only one primary index, you cannot include this clause if you have already created a primary index for a field. Visual FoxPro generates an error if you include more than one PRIMARY KEY clause in CREATE TABLE.

UNIQUE *eExpression3* TAG *TagName3* Creates a candidate index. *eExpression3* specifies any field or combination of fields in the table. However, if you have created a primary index with one of the PRIMARY KEY options, you cannot include the field that was specified for the primary index. TAG *TagName3 s*pecifies a tag name for the candidate index tag that is created. Index tag names can contain up to 10 characters.

A table can have multiple candidate indexes.

FOREIGN KEY *eExpression4* TAG *TagName4* [NODUP] Creates a foreign (non-primary) index, and establishes a relationship to a parent table. *eExpression4* specifies the foreign index key expression and *TagName4* specifies the name of the foreign index key tag that is created. Index tag names can contain up to 10 characters. Include NODUP to create a candidate foreign index.

You can create multiple foreign indexes for the table, but the foreign index expressions must specify different fields in the table.

REFERENCES *TableName3* [TAG *TagName5*] Specifies the parent table to which a persistent relationship is established. Include TAG *TagName5* to establish a relation based on an index tag for the parent table. Index tag names can contain up to 10 characters. If you omit TAG *TagName5,*

the relationship is established using the parent table's primary index key by default.

CHECK *eExpression2* [ERROR *cMessageText2*] Specifies the table validation rule. ERROR *cMessageText2* specifies the error message Visual FoxPro displays when the table validation rule is executed. The message is displayed only when data is changed within a Browse window or Edit window.

FROM ARRAY *ArrayName* Specifies the name of an existing array whose contents are the name, type, precision, and scale for each field in the table. The contents of the array can be defined with the AFIELDS( ) function.

Remarks

The new table is opened in the lowest available work area, and can be accessed by its alias. The new table is opened exclusively, regardless of the current setting of SET EXCLUSIVE.

If a database is open and you don't include the FREE clause, the new table is added to the database. You cannot create a new table with the same name as a table in the database.

If a database is open, CREATE TABLE -SQL requires exclusive use of the database. To open a database for exclusive use, include EXCLUSIVE in OPEN DATABASE.

If a database isn't open when you create the new table, including the NAME, CHECK, DEFAULT, FOREIGN KEY, PRIMARY KEY, or REFERENCES clauses generates an error.

Note that the CREATE TABLE syntax uses commas to separate certain CREATE TABLE options. Also, the NULL, NOT NULL, CHECK, DEFAULT, PRIMARY KEY and UNIQUE clause must be placed within the parentheses containing the column definitions.

Driver Remarks

When your application sends the ODBC SQL statement CREATE TABLE to the data source, the Visual FoxPro ODBC Driver translates the command into the Visual FoxPro CREATE TABLE command using the following syntax:

ODBC Syntax	Visual FoxPro Syntax
CREATE TABLE *base-table-name* (*column-identifier data type* [NOT NULL] [,*column-identifier data type* [NOT NULL] ...)	CREATE TABLE *TableName1* [NAME *LongTableName*] (*FieldName1 FieldType* [(*nFieldWidth* [, *nPrecision*])] [NOT NULL])

When you create a table using the driver, the driver closes the table immediately after creation to allow access to the table by other users. This differs from Visual FoxPro, which leaves the table open exclusively upon creation. However, if a stored procedure on your data source containing a CREATE TABLE statement executes, the table is left open.

If the data source is a database (.DBC file), the Visual FoxPro ODBC Driver creates a table named *LongTableName* with the same name as the *base-table-name*.

Using Data Definition Language (DDL)

You cannot include DDL in the following places:

- In a batch SQL statements that requires a transaction
- Following a previously executed statement that required a transaction if not in autocommit mode and if your application has not yet called SQLTransact.

If you want to create a temporary table, for example, you should create the table before you begin the statement requiring a transaction. If you include the CREATE TABLE statement in a batch SQL statement that requires a transaction, the driver returns an error message.

See Also

ALTER TABLE - SQL

Data Types

INSERT - SQL

SELECT - SQL

DELETE - SQL Command

See Also

Marks records for deletion.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

DELETE FROM [*DatabaseName!*]*TableName*
[WHERE *FilterCondition1* [AND | OR *FilterCondition2* ...]]

Arguments

FROM [*DatabaseName!*]*TableName* Specifies the table in which records are marked for deletion.

DatabaseName! specifies the name of a database containing the table if the containing database is not the database specified with the data source. You must include the name of a database containing the table if the database is not the database specified with the data source. Include the exclamation point (!) delimiter after the database name and before the table name.

WHERE *FilterCondition1* [AND | OR *FilterCondition2* ...] Specifies that Visual FoxPro only mark certain records for deletion.

FilterCondition specifies the criteria that records must meet to be marked for deletion. You can include as many filter conditions as you like, connecting them with the AND or OR operator. You can also use the NOT operator to reverse the value of a logical expression, or use EMPTY( ) to check for an empty field.

Remarks

If SET DELETED is set to ON, records marked for deletion are ignored by all commands that include a scope.

DELETE - SQL uses record locking when marking multiple records for deletion in tables opened for shared access. This reduces record contention in multiuser situations, but may reduce performance. For maximum performance, open the table for exclusive use.

Driver Remarks

When your application sends the ODBC SQL statement DELETE to the data source, the Visual FoxPro ODBC Driver converts the command into the Visual FoxPro DELETE command without translation.

See Also

SET DELETED

DELETE TAG Command

See Also

Removes a tag or tags from a compound index (.CDX) file.

Syntax

DELETE TAG *TagName1* [OF *CDXFileName1*]
[, *TagName2* [OF *CDXFileName2*]] ...

- Or -

DELETE TAG ALL [OF *CDXFileName*]

Arguments

TagName1 OF *CDXFileName1* [, *TagName2* [OF *CDXFileName2*]] ... Specifies a tag to remove from a compound index file. You can delete multiple tags with one DELETE TAG by including a list of tag names separated by commas. If two or more tags with the same name exist in the open index files, you can remove a tag from a specific index file by including OF *CDXFileName*.

ALL [OF *CDXFileName*] Removes every tag from a compound index file. If the current table has a structural compound index file, all tags are removed from the index file, the index file is deleted from the disk, and the flag in the table's header indicating the presence of an associated structural compound index file is removed. Use ALL with OF *CDXFileName* to remove all tags from an open compound index file other than the structural compound index file.

Remarks

Compound index files, created with INDEX, contain tags corresponding to index entries. DELETE TAG is used to remove a tag or tags from open compound index files. You can delete only tags from compound index files open in the current work area. If you remove all the tags from a compound index file, the file is deleted from the disk.

Visual FoxPro looks first for a tag in the structural compound index file (if one is open). If the tag isn't in the structural compound index file, Visual FoxPro then looks for the tag in the other open compound index files.

See Also

INDEX

DROP TABLE Command

Removes a table from the database specified with the data source and deletes it from disk.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

DROP TABLE *TableName* | *FileName* | ?

Settings

TableName Specifies the table to remove from the database specified with the data source and delete from disk.

FileName Specifies a free table to delete from disk.

? Displays the Remove dialog from which you can choose a table to remove from the database specified with the data source and delete from disk.

Remarks

When DROP TABLE is issued, all primary indexes, default values, and validation rules associated with the table are also removed. DROP TABLE also affects other tables in the database specified with the data source if those tables have rules or relations associated with the table being removed. The rules and relations are no longer valid when the table is removed from the database.

Driver Remarks

When your application sends the ODBC SQL statement DROP TABLE to the data source, the Visual FoxPro ODBC Driver converts the command into the Visual FoxPro DROP TABLE command using the following syntax:

ODBC Syntax	Data Source	Visual FoxPro Syntax
DROP TABLE *base-table-name*	Database (.DBC file)	REMOVE TABLE *TableName* DELETE
	Directory of free tables (.DBF files)	ERASE *dbfName* ERASE *cdxName* ERASE *fptName*

INDEX Command

See Also

Creates an index file to display and access table records in a logical order.

Syntax

INDEX ON *eExpression* TO *IDXFileName* | TAG *TagName* [OF *CDXFileName*]
[FOR *lExpression*]
[COMPACT]
[ASCENDING | DESCENDING]
[UNIQUE | CANDIDATE]
[ADDITIVE]

Arguments

eExpression Specifies an index expression that can include the name of a field or fields from the current table. An index key based on the index expression is created in the index file for each record in the table. Visual FoxPro uses these keys to display and access records in the table.

Note Although not recommended, *eExpression* can also be a memory variable, an array element, or a field or field expression from a table in another work area. Memo fields cannot be used alone in index file expressions; they must be combined with other character expressions. If you access an index that contains a variable or field that no longer exists or cannot be located, Visual FoxPro generates an error message.

If you attempt to build an index with a key that varies in length, the key will be padded with spaces. Variable-length index keys aren't supported in Visual FoxPro.

It is possible to create an index key with zero length. For example, a zero length index key is created when the index expression is a substring of an empty memo field. A zero length index key generates an error message. When Visual FoxPro creates an index, it evaluates fields in the first record in the table. If a field is empty, it may be necessary to enter some temporary data in the field in the first record to prevent a 0 length index key.

TO *IDXFileName* Creates an .IDX index file. The index file is given the default extension .IDX.

TAG *TagName* [OF *CDXFileName*] Creates a compound index file. A compound index file is a single index file that consists of any number of separate tags (index entries). Each tag is identified by its unique tag name. Tag names must begin with a letter or an underscore and can consist of any combination of up to 10 letters, digits, or underscores. The number of tags in a compound index file is limited only by available memory and disk space.

Multiple-entry compound index files are always compact. It isn't necessary to include COMPACT when creating a compound index file. Names of compound index files are given a .CDX extension.

Two types of compound index files can be created: structural and non-structural.

Structural Compound Index Files You can create a structural compound index file with TAG *TagName* by excluding the optional OF *CDXFileName* clause. A structural compound index file always has the same base name as the table and is automatically opened when the table is opened.

Non-Structural Compound Index Files You can create a non-structural compound index file by including OF *CDXFileName* after TAG *TagName*. Unlike a structural compound index file, a non-structural compound index file must be explicitly opened with the INDEX clause in USE.

If a compound index file has already been created and opened, issuing INDEX with TAG *TagName* adds a tag to the compound index file.

FOR *lExpression* Specifies a condition whereby only records that satisfy the filter expression *lExpression* are available for display and access; index keys are created in the index file for just those records matching the filter expression.

Rushmore optimizes an INDEX ... FOR *lExpression* command if *lExpression* is an optimizable

expression. For best performance, use an optimizable expression in the FOR clause.

COMPACT Creates a compact .IDX file.

ASCENDING Specifies an ascending order for the .CDX file. By default, .CDX tags are created in ascending order (you can include ASCENDING as a reminder of the index file's order). A table can be indexed in reverse order by including DESCENDING.

DESCENDING Specifies a descending order for the .CDX file. You can't include DESCENDING when creating .IDX index files.

UNIQUE Specifies that only the first record encountered with a particular index key value is included in an .IDX file or a .CDX tag. UNIQUE can be used to prevent the display of or access to duplicate records. All records added with duplicate index keys are excluded from the index file. Using the UNIQUE option of INDEX is identical to executing SET UNIQUE ON before issuing INDEX or REINDEX.

When a UNIQUE index or index tag is active and a duplicate record is changed in a manner that changes its index key, the index or index tag is updated. However, the next duplicate record with the original index key cannot be accessed or displayed until you reindex the file using REINDEX.

CANDIDATE Creates a candidate structural index tag. The CANDIDATE keyword can be included only when creating a structural index tag; otherwise Visual FoxPro generates an error message.

A candidate index tag prevents duplicate values in the field or combination of fields specified in the index expression *eExpression*. The term "candidate" refers to the type of index; because candidate indexes prevent duplicate values, they qualify as a "candidate" to be a primary index.

Visual FoxPro generates an error if you create a candidate index tag for a field or combination of fields that already contain duplicate values.

ADDITIVE Keeps open any previously opened index files. If you omit the ADDITIVE clause when you create an index file or files for a table with INDEX, any previously opened index files (except the structural compound index) are closed.

Remarks

Records in a table that has an index file are displayed and accessed in the order specified by the index expression. The physical order of the records in the table isn't changed by an index file.

Index Types

Visual FoxPro lets you create two types of index files:

- Compound .CDX index files containing multiple index entries called tags
- .IDX index files containing one index entry

You can also create a structural compound index file, which is automatically opened with the table.

Tip Because structural compound index files are automatically opened when the table is opened, they are the preferred index type.

Include COMPACT to create compact .IDX index files. Compound index files are always compact.

Index Order and Updating

Only one index file (the master index file) or tag (the master tag) controls the order in which the table is displayed or accessed. Certain commands (SEEK, for example) use the master index file or tag to search for records. However, all open .IDX and .CDX index files are updated as changes are made to the table.

User-Defined Functions

Although an index expression can contain a user-defined function, you should not use user-defined functions in an index expression. User-defined function in an index expression increase the time it takes to create or update the index. Also, index updates may not occur when a user-defined function

is used for an index expression.

If you use a user-defined function in an index expression, Visual FoxPro must be able to locate the user-defined function. When Visual FoxPro creates an index, the index expression is saved in the index file, but only a reference to the user-defined function is included in the index expression.

See Also

ALTER TABLE

DELETE TAG

SET COLLATE

SET UNIQUE

INSERT - SQL Command

See Also

Appends a record to the end of a table that contains the specified field values.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

INSERT INTO *dbf_name* [(*fname1* [, *fname2*, ...])]
VALUES (*eExpression1* [, *eExpression2*, ...])

Arguments

INSERT INTO *dbf_name* Specifies the name of the table to which the new record is appended. *dbf_name* can include a path and can be a name expression.

If the table you specify isn't open, it is opened exclusively in a new work area and the new record is appended to the table. The new work area isn't selected; the current work area remains selected.

If the table you specify is open, INSERT appends the new record to the table. If the table is open in a work area other than the current work area, it isn't selected after the record is appended; the current work area remains selected.

[(*fname1* [, *fname2* [, ...]])] Specifies the names of the fields in the new record into which the values are inserted.

VALUES (*eExpression1* [, *eExpression2* [, ...]]) Specifies the field values inserted into the new record. If you omit the field names, you must specify the field values in the order defined by the table structure.

Remarks

The new record contains the data listed in the VALUES clause.

Driver Remarks

When your application sends the ODBC SQL statement INSERT to the data source, the Visual FoxPro ODBC Driver converts the command into the Visual FoxPro INSERT command without translation.

See Also

CREATE TABLE - SQL

SELECT - SQL

SELECT - SQL Command

See Also

Retrieves data from one or more tables.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

SELECT [ALL | DISTINCT]
[*Alias*.] *Select_Item* [AS *Column_Name*]
[, [*Alias*.] *Select_Item* [AS *Column_Name*] ...]
FROM [*DatabaseName!*]*Table* [*Local_Alias*]
[, [*DatabaseName!*]*Table* [*Local_Alias*] ...]
[WHERE *JoinCondition* [AND *JoinCondition*
…]
[AND | OR *FilterCondition* [AND | OR *FilterCondition* ...]]]]
[GROUP BY *GroupColumn* [, *GroupColumn* ...]]
[HAVING *FilterCondition*]
[UNION [ALL] *SELECTCommand*]
[ORDER BY *Order_Item* [ASC | DESC] [, *Order_Item* [ASC | DESC] ...]]

Arguments

[ALL | DISTINCT]
[*Alias*.] *Select_Item* [AS *Column_Name*]
[, [*Alias*.] *Select_Item* [AS *Column_Name*] ...] The SELECT clause specifies the fields, constants, and expressions that are displayed in the query results.

ALL, by default, displays all the rows in the query results.

DISTINCT excludes duplicates of any rows from the query results.

Note You can use DISTINCT only once per SELECT clause.

Alias. qualifies matching item names. Each item you specify with *Select_Item* generates one column of the query results. If two or more items have the same name, include the table alias and a period before the item name to prevent columns from being duplicated.

Select_Item specifies an item to be included in the query results. An item can be one of the following:

- The name of a field from a table in the FROM clause.
- A constant specifying that the same constant value is to appear in every row of the query results.
- An expression that can be the name of a user-defined function.

User-Defined Functions with SELECT

Although using user-defined functions in the SELECT clause has obvious benefits, you should also consider the following restrictions:

- The speed of operations performed with SELECT may be limited by the speed at which such user-defined functions are executed. High-volume manipulations involving user-defined functions may be better accomplished by using API and user-defined functions written in C or assembly language.
- The only reliable way to pass values to user-defined functions invoked from SELECT is by the argument list passed to the function when it is invoked.
- If you experiment and discover a supposedly forbidden manipulation that works correctly in a certain version of FoxPro, there is no guarantee it will continue to work in later versions.

Apart from these restrictions, user-defined functions are acceptable in the SELECT clause. However, don't forget that using SELECT might slow performance.

The following field functions are available for use with a select item that is a field or an expression involving a field:

- AVG(*Select_Item*), which averages a column of numeric data.
- COUNT(*Select_Item*), which counts the number of select items in a column. COUNT(*) counts the number of rows in the query output.
- MIN(*Select_Item*), which determines the smallest value of *Select_Item* in a column.
- MAX(*Select_Item*), which determines the largest value of *Select_Item* in a column.
- SUM(*Select_Item*), which totals a column of numeric data.

You cannot nest field functions.

AS *Column_Name* Specifies the heading for a column in the query output. This is useful when *Select_Item* is an expression or contains a field function and you want to give the column a meaningful name. *Column_Name* can be an expression but cannot contain characters (for example, spaces) that aren't permitted in table field names.

FROM [*DatabaseName!*]*Table* [*Local_Alias*]
[, [*DatabaseName!*]*Table* [*Local_Alias*] ...] Lists the tables containing the data that the query retrieves. If no table is open, Visual FoxPro displays the Open dialog box so you can specify the file location. Once open, the table remains open once the query is complete.

DatabaseName! specifies the name of a database other than the one specified with the data source. You must include the name of database containing the table if the database is not specified with the data source. Include the exclamation point (!) delimiter after the database name and before the table name.

Local_Alias specifies a temporary name for the table named in *Table*. If you specify a local alias, you must use the local alias in place of the table name throughout the SELECT statement. The local alias doesn't affect the Visual FoxPro environment.

WHERE *JoinCondition* [AND *JoinCondition* ...]
[AND | OR *FilterCondition* [AND | OR *FilterCondition* ...]] Tells Visual FoxPro to include only certain records in the query results. WHERE is required to retrieve data from multiple tables.

JoinCondition specifies fields that link the tables in the FROM clause. If you include more than one table in a query, you should specify a join condition for every table after the first.

Important Keep the following information in mind when creating join conditions:

- If you include two tables in a query and don't specify a join condition, every record in the first table is joined with every record in the second table as long as the filter conditions are met. Such a query can produce lengthy results.
- Use caution when joining tables with empty fields because Visual FoxPro matches empty fields. For example, if you join on CUSTOMER.ZIP and INVOICE.ZIP, and CUSTOMER contains 100 empty zip codes and INVOICE contains 400 empty zip codes, the query output contains 40,000 extra records resulting from the empty fields. Use the EMPTY( ) function to eliminate empty records from the query output.

You must use the AND operator to connect multiple join conditions. Each join condition has the following form:

FieldName1 Comparison FieldName2

FieldName1 is the name of a field from one table, *FieldName2* is the name of a field from another table, and *Comparison* is one of the following operators.

Operator	Comparison
=	Equal
==	Exactly equal
LIKE	SQL LIKE

<>, !=, #	Not equal
>	More than
>=	More than or equal to
<	Less than
<=	Less than or equal to

When you use the = operator with strings, it acts differently depending on the setting of SET ANSI. When SET ANSI is set to OFF, Visual FoxPro treats string comparisons in a manner familiar to Xbase users. When SET ANSI is set to ON, Visual FoxPro follows ANSI standards for string comparisons. See SET ANSI and SET EXACT for additional information about how Visual FoxPro performs string comparisons.

FilterCondition specifies the criteria that records must meet to be included in the query results. You can include as many filter conditions as you like in a query, connecting them with the AND or OR operator. You can also use the NOT operator to reverse the value of a logical expression, or use EMPTY( ) to check for an empty field. *FilterCondition* can take any of the forms in the following examples:

Example 1 *FieldName1 Comparison FieldName2*

```
customer.cust_id = orders.cust_id
```

Example 2 *FieldName Comparison Expression*

```
payments.amount >= 1000
```

Example 3 *FieldName Comparison* ALL (*Subquery*)

```
company < ALL ;
(SELECT company FROM customer WHERE country = "UK")
```

When the filter condition includes ALL, the field must meet the comparison condition for all values generated by the subquery before its record is included in the query results.

Example 4 *FieldName Comparison* ANY | SOME (*Subquery*)

```
company < ANY ;
(SELECT company FROM customer WHERE country = "UK")
```

When the filter condition includes ANY or SOME, the field must meet the comparison condition for at least one of the values generated by the subquery.

The following example checks to see whether the values in the field are within a specified range of values.

Example 5 *FieldName* [NOT] BETWEEN *Start_Range* AND *End_Range*

```
customer.postalcode BETWEEN 90000 AND 99999
```

The following example checks to see whether at least one row meets the criteria in the subquery. When the filter condition includes EXISTS, the filter condition evaluates to true (.T.) unless the subquery evaluates to the empty set.

Example 6 [NOT] EXISTS (*Subquery*)

```
EXISTS ;
(SELECT * FROM orders WHERE customer.postalcode =
   orders.postalcode)
```

Example 7 *FieldName* [NOT] IN *Value_Set*

```
customer.postalcode NOT IN ("98052","98072","98034")
```

When the filter condition includes IN, the field must contain one of the values before its record is included in the query results.

Example 8 *FieldName* [NOT] IN (*Subquery*)

```
customer.cust_id IN ;
(SELECT orders.cust_id FROM orders WHERE orders.city="Seattle")
```

Here, the field must contain one of the values returned by the subquery before its record is included in the query results.

Example 9 *FieldName* [NOT] LIKE *cExpression*

```
customer.country NOT LIKE "UK"
```

This filter condition searches for each field that matches *cExpression*. You can use the percent sign (%) and underscore (_) wildcards as part of *cExpression*. The underscore represents a single unknown character in the string.

GROUP BY *GroupColumn* [, *GroupColumn* ...] Groups rows in the query based on values in one or more columns. *GroupColumn* can be the name of a regular table field, or a field that includes a SQL field function, or a numeric expression indicating the location of the column in the result table (the leftmost column number is 1).

HAVING *FilterCondition* Specifies a filter condition which groups must meet to be included in the query results. HAVING should be used with GROUP BY. It can include as many filter conditions as you like, connected with the AND or OR operator. You can also use NOT to reverse the value of a logical expression.

FilterCondition cannot contain a subquery.

A HAVING clause without a GROUP BY clause acts like a WHERE clause. You can use local aliases and field functions in the HAVING clause. Use a WHERE clause for faster performance if your HAVING clause contains no field functions.

[UNION [ALL] *SELECTCommand*] Combines the final results of one SELECT with the final results of another SELECT. By default, UNION checks the combined results and eliminates duplicate rows. Use parentheses to combine multiple UNION clauses.

ALL prevents UNION from eliminating duplicate rows from the combined results.

UNION clauses follow these rules:

- You cannot use UNION to combine subqueries.
- Both SELECT commands must have the same number of columns in their query output.
- Each column in the query results of one SELECT must have the same data type and width as the corresponding column in the other SELECT.
- Only the final SELECT can have an ORDER BY clause, which must refer to output columns by number. If an ORDER BY clause is included, it affects the entire result.

You can also use the UNION clause to simulate an outer join.

When you join two tables in a query, only records with matching values in the joining fields are included in the output. If a record in the parent table doesn't have a corresponding record in the child table, the record in the parent table isn't included in the output. An outer join allows you to include all the records in the parent table in the output, along with the matching records in the child table. To create an outer join in Visual FoxPro, you need to use a nested SELECT command, as in the following example:

```
SELECT customer.company, orders.order_id, orders.emp_id ;
   FROM customer, orders ;
   WHERE customer.cust_id = orders.cust_id ;
UNION ;
   SELECT customer.company, 0, 0 ;
   FROM customer ;
   WHERE customer.cust_id NOT IN ;
   (SELECT orders.cust_id FROM orders)
```

Note Be sure to include the space immediately preceding each semicolon. Otherwise, you'll receive an error.

The section of the command before the UNION clause selects records from both tables that have matching values. The customer companies that do not have associated invoices are not included. The section of the command after the UNION clause selects records in the `customer` table that do

not have matching records in the `orders` table.

Regarding the second section of the command, note the following:

- The SELECT statement within the parentheses is processed first. This statement results in a selection of all customer numbers in the `orders` table.
- The WHERE clause finds all customer numbers in the `customer` table that are not in the `orders` table. Since the first section of the command provided all companies that had a customer number in the `orders` table, all companies in the `customer` table are now included in the query results.
- Because the structures of tables included in a UNION must be identical, there are two placeholders in the second SELECT statement to represent `orders.order_id` and `orders.emp_id` from the first SELECT statement.

Note The placeholders must be the same type as the fields they represent. If the field is a date type, the placeholder should be { / / }. If the field is a character field, the placeholder should be the empty string, ("").

ORDER BY *Order_Item* [ASC | DESC] [, *Order_Item* [ASC | DESC] ...] Sorts the query results based on the data in one or more columns. Each *Order_Item* must correspond to a column in the query results and can be one of the following:

- A field in a FROM table that is also a select item in the main SELECT clause (not in a subquery).
- A numeric expression indicating the location of the column in the result table. (The leftmost column is number 1.)

ASC specifies an ascending order for query results, according to the order item or items, and is the default for ORDER BY.

DESC specifies a descending order for query results.

Query results appear unordered if you don't specify an order with ORDER BY.

Remarks

SELECT is a SQL command that is built into Visual FoxPro like any other Visual FoxPro command. When you use SELECT to pose a query, Visual FoxPro interprets the query and retrieves the specified data from the tables. You can create a SELECT query from within:

- The Command window
- A Visual FoxPro program (as with any other Visual FoxPro command)

Note that SELECT does not respect the current filter condition specified with SET FILTER.

Note A *subquery*, referred to in the following arguments, is a SELECT within a SELECT and must be enclosed in parentheses. You can have up to two subqueries at the same level (not nested) in the WHERE clause (see that section of the arguments). Subqueries can contain multiple join conditions.

Driver Remarks

When your application sends the ODBC SQL statement SELECT to the data source, the Visual FoxPro ODBC Driver converts the command into the Visual FoxPro SELECT command without translation unless the command contains an ODBC escape sequence. Items enclosed in an ODBC escape sequence are converted to Visual FoxPro syntax. For more information on using ODBC escape sequences, see Time and Date Functions and the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*.

See Also

CREATE TABLE - SQL

INSERT - SQL

SET ANSI

SET EXACT

SET ANSI Command

See Also

Determines how comparisons between strings of different lengths are made with the = operator in Visual FoxPro SQL commands.

Syntax

SET ANSI ON | OFF

Arguments

ON (Default for the driver; the default for Visual FoxPro is OFF) Pads the shorter string with the blanks needed to make it equal to the longer string's length. The two strings are then compared character for character for their entire lengths. Consider this comparison:

```
'Tommy' = 'Tom'
```

The result is false (.F.) if SET ANSI is on because when padded, 'Tom' becomes 'Tom ' and the strings 'Tom ' and 'Tommy' don't match character for character.

The == operator uses this method for comparisons in Visual FoxPro SQL commands.

OFF Specifies that the shorter string not be padded with blanks. The two strings are compared character for character until the end of the shorter string is reached. Consider this comparison:

```
'Tommy' = 'Tom'
```

The result is true (.T.) when SET ANSI is off because the comparison stops after 'Tom'.

Remarks

SET ANSI determines whether the shorter of two strings is padded with blanks when a SQL string comparison is made. SET ANSI has no effect on the == operator; when you use the == operator, the shorter string is always padded with blanks for the comparison.

String Order

In SQL commands, the left-to-right order of the two strings in a comparison is irrelevant—switching a string from one side of the = or == operator to the other doesn't affect the result of the comparison.

See Also

SELECT - SQL

SET EXACT

SET BLOCKSIZE Command

Specifies how disk space is allocated for the storage of memo fields.

Syntax

SET BLOCKSIZE TO *nBytes*

Arguments

nBytes Specifies the block size in which disk space for memo fields is allocated. If *nBytes* is 0, disk space is allocated in single bytes (blocks of 1 byte). If *nBytes* is an integer between 1 and 32, disk space is allocated in blocks of *nBytes* bytes multiplied by 512. If *nBytes* is greater than 32, disk space is allocated in blocks of *nBytes* bytes. If you specify a block size value greater than 32, you can save substantial disk space.

Remarks

The default value for SET BLOCKSIZE is 64. To reset the block size to a different value after the file has been created, set it to a new value and then use COPY to create a new table. The new table has the specified block size.

SET COLLATE Command

See Also

Specifies a collation sequence for character fields in subsequent indexing and sorting operations.

Syntax

SET COLLATE TO *cSequenceName*

Arguments

cSequenceName Specifies a collation sequence. The following collation sequence options are available.

Options	Language
DUTCH	Dutch
GENERAL	English, French, German, Modern Spanish, Portuguese, and other Western European languages
GERMAN	German phone book order (DIN)
ICELAND	Icelandic
MACHINE	Machine (the default collation sequence for earlier FoxPro versions)
NORDAN	Norwegian, Danish
SPANISH	Traditional Spanish
SWEFIN	Swedish, Finnish
UNIQWT	Unique Weight

Note When you specify the SPANISH option, "ch" is a single letter that sorts between "c" and "d", and "ll" sorts between "l" and "m".

If you specify a collation sequence option as a literal character string, be sure to enclose the option in quotation marks:

```
SET COLLATE TO "SWEFIN"
```

MACHINE is the default collation sequence option and is the sequence Xbase users are familiar with. Characters are ordered as they appear in the current code page.

GENERAL may be preferable for U.S. and Western European users. Characters are ordered as they appear in the current code page. In FoxPro versions earlier than 2.5, indexes might have been created using the UPPER( ) or LOWER( ) functions to convert character fields to a consistent case. In FoxPro versions later than 2.5, you can instead specify the GENERAL collation sequence option and omit the UPPER( ) conversion.

Note that if you specify a collation sequence option other than MACHINE and if you create an .IDX file, a compact .IDX is always created.

Use SET("COLLATE") to return the current collation sequence.

You can specify a collating sequence for a data source by using the ODBC Visual FoxPro Setup Dialog Box or by using the Collate keyword in your connection string with **SQLDriverConnect.**. This is identical to issuing the following command:

```
SET COLLATE TO cSequenceName
```

Remarks

SET COLLATE enables you to order tables containing accented characters for any of the supported languages. Changing the setting of SET COLLATE doesn't affect the collating sequence of previously opened indexes. Visual FoxPro automatically maintains existing indexes, providing the flexibility to create many different types of indexes, even for the same field.

For example, if an index is created with SET COLLATE set to GENERAL, and the SET COLLATE setting is later changed to SPANISH, the index retains the GENERAL collation sequence.

See Also

ODBC Visual FoxPro Setup Dialog Box

SET DELETED Command

See Also

Specifies whether records marked for deletion are processed and whether they are available for use in other commands.

Syntax

SET DELETED ON | OFF

Arguments

ON (Default for the driver; the default for Visual FoxPro is OFF) Specifies that commands which operate on records (including records in related tables) using a scope ignore records marked for deletion.

OFF Specifies that records marked for deletion can be accessed by commands that operate on records (including records in related tables) using a scope.

Remarks

Queries that use DELETED( ) to test the status of records can be optimized using Rushmore technology if the table is indexed on DELETED( ).

Important SET DELETED is ignored if the default scope for the command is the current record or you include a scope of a single record. INDEX always ignores SET DELETED and indexes all records in the table.

See Also

DELETE - SQL

SET EXACT Command

See Also

Specifies the rules for comparing two strings of different lengths.

Syntax

SET EXACT ON | OFF

Arguments

ON Specifies that expressions must match character for character to be equivalent. Any trailing blanks in the expressions are ignored for the comparison. For the comparison, the shorter of the two expressions is padded on the right with blanks to match the length of the longer expression.

OFF (Default) Specifies that, to be equivalent, expressions must match character for character until the end of the expression on the right side is reached.

Remarks

The SET EXACT setting has no effect if both strings are the same length.

String Comparisons

Visual FoxPro has two relational operators that test for equality.

The = operator performs a comparison between two values of the same type. This operator is suited for comparing character, numeric, date, and logical data.

However, when you compare character expressions with the = operator, the results might not be exactly what you expect. Character expressions are compared character for character from left to right until one of the expressions isn't equal to the other, or until the end of the expression on the right side of the = operator is reached (SET EXACT OFF), or until the ends of both expressions are reached (SET EXACT ON).

The == operator can be used when an exact comparison of character data is needed. If two character expressions are compared with the == operator, the expressions on both sides of the == operator must contain exactly the same characters, including blanks, to be considered equal. The SET EXACT setting is ignored when character strings are compared using ==.

The following table shows how the choice of operator and the SET EXACT setting affect comparisons. (An underscore represents a blank space.)

Comparison	= EXACT OFF	= EXACT ON	== EXACT ON or OFF
"abc" = "abc"	Match	Match	Match
"ab" = "abc"	No match	No match	No match
"abc" = "ab"	Match	No match	No match
"abc" = "ab_"	No match	No match	No match
"ab" = "ab_"	No match	Match	No match
"ab_" = "ab"	Match	Match	No match
"" = "ab"	No match	No match	No match
"ab" = ""	Match	No match	No match
"__" = ""	Match	Match	No match
"" = "___"	No match	Match	No match
TRIM("___") = ""	Match	Match	Match
"" = TRIM("___")	Match	Match	Match

See Also

SET ANSI

SET EXCLUSIVE Command

See Also

Specifies whether table files are opened for exclusive or shared use on a network.

Syntax

SET EXCLUSIVE ON | OFF

Arguments

ON Limits accessibility of a table opened on a network to the user who opened it. The table isn't accessible to other users on the network. SET EXCLUSIVE ON also prevents all other users from having read-only access.

OFF (Default for the driver; the defaults for Visual FoxPro are ON for the global data session and OFF for a private data session.) Allows a table opened on a network to be shared and modified by any user on the network.

Remarks

Changing the setting of SET EXCLUSIVE doesn't change the status of previously opened tables. For example, if a table is opened with SET EXCLUSIVE set to ON, and SET EXCLUSIVE is later changed to OFF, the table retains its exclusive-use status.

See Also

ODBC Visual FoxPro Setup Dialog Box

SET NULL Command

See Also

Determines how null values are supported by the ALTER TABLE - SQL, CREATE TABLE - SQL and INSERT - SQL commands.

Syntax

SET NULL ON | OFF

Arguments

ON (Default for the driver; the default for Visual FoxPro is OFF) Specifies that all columns in a table created with ALTER TABLE and CREATE TABLE will allow null values. You can override null value support for columns in the table by including the NOT NULL clause in the columns' definitions.

Also, specifies that INSERT - SQL will insert null values into any columns not included in the INSERT - SQL VALUE clause. INSERT - SQL will only insert null values into columns that allow null values.

OFF Specifies that all columns in a table created with ALTER TABLE and CREATE TABLE will not allow null values. You can designate null value support for columns in ALTER TABLE and CREATE TABLE by including the NULL clause in the columns' definitions.

Also, specifies that INSERT - SQL will insert blank values into any columns not included in the INSERT - SQL VALUE clause.

Remarks

SET NULL only affects how null values are supported by ALTER TABLE, CREATE TABLE and INSERT - SQL. Other commands are unaffected by SET NULL.

See Also

ALTER TABLE

CREATE TABLE - SQL

INSERT - SQL

SET PATH Command

See Also

Specifies a path for file searches. For driver-specific information, see Driver Remarks below.

Syntax

SET PATH TO [*Path*]

Arguments

TO [*Path*] Specifies the directories you want Visual FoxPro to search. Use commas or semicolons to separate the directories.

Remarks

SET PATH allows you to specify search paths for other Visual FoxPro programs that can be called within stored procedures. SET PATH will not change the path of the data source that you've specified for the connection.

Issue SET PATH TO without *Path* to restore the path to the default directory or folder.

Driver Remarks

If you issue SET PATH in a stored procedure, it will be ignored by the following functions and commands:

- Catalog functions such as **SQLTables** and **SQLColumns** will ignore the new path and continue to reference the path specified by the data source in **SQLPrepare** or **SQLExecDirect**.
- Commands such as SELECT, INSERT, UPDATE, DELETE, and CREATE TABLE will ignore the new path and continue to reference the path specified by the data source in **SQLPrepare** or **SQLExecDirect**.

If you issue SET PATH in a stored procedure and don't subsequently set the path back to its original state, other connections to the database will use the new path (because SET PATH is not scoped to data sessions).

If you want to create or select or update tables in a directory other than that specified by the data source, specify the full path of the file with your command.

See Also

ODBC Visual FoxPro Setup Dialog Box

SQLColumns

SQLDriverConnect

SQLTables

SET REPROCESS Command

Specifies how many times or for how long to lock a file or record after an unsuccessful locking attempt.

Syntax

SET REPROCESS TO *nAttempts* [SECONDS] | TO AUTOMATIC

Arguments

TO *nAttempts* [SECONDS] Specifies the number of times or number of seconds to try to lock a record or file after an initial unsuccessful attempt. The default value is 0, the maximum value is 32,000.

SECONDS specifies that Visual FoxPro attempts to lock a file or record for *nAttempts* seconds. It's available only when *nAttempts* is greater than zero.

For example, if *nAttempts* is 30, Visual FoxPro attempts to lock a record or file up to 30 times. If you also include SECONDS (`SET REPROCESS TO 30 SECONDS`), Visual FoxPro continuously attempts to lock a record or file for up to 30 seconds.

If an ON ERROR routine is in effect, and attempts by a command to lock the record or file are unsuccessful, the ON ERROR routine is executed. However, if a function attempts the lock, an ON ERROR routine isn't executed and the function returns false (.F.).

If an ON ERROR routine isn't in effect, a command attempts to lock the record or file, and the lock can't be placed, an error is generated. If a function attempts to place the lock, the alert isn't displayed, and the function returns false (.F.).

If *nAttempts* is 0 (the default value), and you issue a command or function that attempts to lock a record or file, Visual FoxPro tries to lock the record or file indefinitely. The lock is placed and the system message is cleared if the record or file becomes available for locking while you wait. If a function attempted to place the lock, the function returns true (.T.).

If an ON ERROR routine is in effect and a command is attempting to lock the record or file, the ON ERROR routine takes precedence over additional attempts to lock the record or file. The ON ERROR routine is immediately executed. Visual FoxPro does not attempt additional record or file locks and does not display the system message.

If *nAttempts* is -1, Visual FoxPro attempts to lock the record or file indefinitely, and an ON ERROR routine isn't executed.

If a lock has been placed by another user on the record or file you are attempting to lock, you must wait until the user releases the lock.

TO AUTOMATIC Specifies that Visual FoxPro attempts to lock the record or file indefinitely. (`SET REPROCESS TO -2` is an equivalent command.)

Remarks

The first attempt to lock a record or file isn't always successful. Frequently, a record or file is locked by another user on the network. SET REPROCESS determines if Visual FoxPro makes additional attempts to lock the record or file when the initial attempt is unsuccessful. You can specify either how many times additional attempts are made or for how long the attempts are made. An ON ERROR routine affects how unsuccessful lock attempts are handled.

SET UNIQUE Command

See Also

Specifies if records with duplicate index key values are maintained in an index file.

Syntax

SET UNIQUE ON | OFF

Arguments

ON Specifies that any record with a duplicate index key value not be included in the index file. Only the first record with the original index key value is included in the index file.

OFF (Default) Specifies that records with duplicate index key values be included in the index file.

Remarks

An index file retains its SET UNIQUE setting when you issue REINDEX. For more information, see INDEX.

See Also

INDEX

UPDATE - SQL Command

See Also

Updates records in a table with new values.

The Visual FoxPro ODBC Driver supports the native Visual FoxPro language syntax for this command. For driver-specific information, see Driver Remarks below.

Syntax

UPDATE [*DatabaseName1!*]*TableName1*
SET *Column_Name1 = eExpression1*
[, *Column_Name2 = eExpression2* ...]
WHERE *FilterCondition1* [AND | OR *FilterCondition2* ...]]

Arguments

UPDATE [*DatabaseName1!*]*TableName1* Specifies the table in which records are updated with new values.

DatabaseName1! specifies the name of a database other than the database specified with the data source containing the table. You must include the name of the database containing the table if the database is not the current one. Include the exclamation point (!) delimiter after the database name and before the table name.

SET *Column_Name1 = eExpression1*
[, *Column_Name2 = eExpression2* Specifies the columns that are updated and their new values. If you omit the WHERE clause, every row in the column is updated with the same value.

WHERE *FilterCondition1* [AND | OR *FilterCondition2* ...]] Specifies the records that are updated with new values.

FilterCondition specifies the criteria that records must meet to be updated with new values. You can include as many filter conditions as you like, connecting them with the AND or OR operator. You can also use the NOT operator to reverse the value of a logical expression, or use EMPTY( ) to check for an empty field.

Remarks

UPDATE - SQL can only update records in a single table.

Unlike REPLACE, UPDATE - SQL uses record locking when updating multiple records in tables opened for shared access. This reduces record contention in multiuser situations, but may reduce performance. For maximum performance, open the table for exclusive use or use FLOCK( ) to lock the table.

Driver Remarks

When your application sends the ODBC SQL statement UPDATE to the data source, the Visual FoxPro ODBC Driver converts the command into the Visual FoxPro UPDATE command without translation.

See Also

DELETE - SQL

INSERT - SQL

Visual FoxPro Field Data Types

The following table lists the values for the *FieldType* argument in ALTER TABLE and CREATE TABLE and indicates whether *nFieldWidth* and *nPrecision* arguments are required.

FieldType	*nFieldWidth*	*nPrecision*	Description
C	n	-	Character field of width *n*
D	-	-	Date
T	-	-	DateTime
N	n	d	Numeric field of width *n* with *d* decimal places
F	n	d	Floating numeric field of width *n* with *d* decimal places
I	-	-	Integer
B	-	d	Double
Y	-	-	Currency
L	-	-	Logical
M	-	-	Memo
G	-	-	General