



Microsoft® Data Access Components 2.5 SDK

PART III – Microsoft ODBC API Specification

Installing and Configuring ODBC Software

This PDF file contains contents from the Microsoft® Data Access SDK online *ODBC Programmer's Reference* at the following web site:

<http://msdn.microsoft.com/isapi/msdnlib2.idc?theURL=/library/psdk/dasdk/odin8w4s.htm>

For your convenience, the main sections of the Microsoft documentation are provided in three PDF files that are available on the SOLID Web site:

- The Part I Microsoft ODBC API Specification PDF file contains introductory information on ODBC and information on developing applications and drivers.
- The Part II Microsoft ODBC API Specification PDF file contains the ODBC API Reference, descriptions of each ODBC function.
- The Part III Microsoft ODBC API Specification PDF file contains information on installing and configuring ODBC software, and setup, installer, and translation DLL functions.

NOTE: Refer to Chapter 2, "Using SOLID ODBC API," in the **SOLID Programmer Guide** for SOLID-specific usage. This PDF file also refers you to SOLID manuals for information on SOLID usage and to the Microsoft Web site (noted above) for those portions of the documentation that are not included in the PDF file.

Copyright © 2000 Microsoft Corporation. All rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Solid Information Technology.

Contents

| | |
|---|-----------|
| About Part III: Installing and Configuring ODBC Software | 4 |
| Section 1: Installing and Configuring ODBC Software | 5 |
| Chapter 18: Installing ODBC Components | 6 |
| Setup Program | 7 |
| Installer DLL | 8 |
| Driver Setup DLL | 9 |
| Usage Counting | 10 |
| Redistributable Files | 12 |
| Registry Entries for ODBC Components | 13 |
| Chapter 19: Configuring Data Sources | 22 |
| Configuration Components | 23 |
| Registry Entries for Data Sources | 28 |
| Section 2: DLL Functions | 33 |
| Chapter 21: Installer DLL API Function Summary | 34 |
| Chapter 22: Setup DLL API Reference | 35 |
| ConfigDriver | 36 |
| ConfigDSN | 39 |
| ConfigTranslator | 43 |
| Chapter 23: Installer DLL API Reference | 45 |
| SQLConfigDataSource | 46 |
| SQLConfigDriver | 49 |
| SQLCreateDataSource | 52 |
| SQLGetConfigMode | 58 |
| SQLGetInstalledDrivers | 60 |
| SQLGetTranslator | 62 |
| SQLGetPrivateProfileString | 65 |
| SQLInstallDriverEx | 67 |
| SQLInstallDriverManager | 71 |
| SQLInstallerError | 73 |
| SQLInstallTranslator | 75 |
| SQLInstallTranslatorEx | 76 |
| SQLManageDataSources | 79 |
| SQLPostInstallerError | 85 |
| SQLReadFileDSN | 86 |
| SQLRemoveDefaultDataSource | 89 |
| SQLRemoveDriver | 90 |
| SQLRemoveDriverManager | 93 |
| SQLRemoveDSNFromIni | 95 |
| SQLRemoveTranslator | 97 |
| SQLSetConfigMode | 99 |
| SQLValidDSN | 101 |
| SQLWriteDSNToIni | 103 |
| SQLWriteFileDSN | 105 |
| SQLWritePrivateProfileString | 107 |

| | |
|---|------------|
| Chapter 24: Translation DLL Function Reference | 109 |
| SQLDriverToDataSource | 110 |
| SQLDataSourceToDriver | 113 |

About Part III: Installing and Configuring ODBC Software

This PDF file is an excerpt from the Microsoft *ODBC Programmer's Reference* and is divided into the following parts:

[Section 1: Installing and Configuring ODBC](#) which describes how to install, remove, and configure ODBC components.

[Section 2: DLL Functions](#), which describes Installer, Setup, and Translator DLL functions.

Section 1: Installing and Configuring ODBC Software

Section 1 of the *ODBC Programmer's Reference* contains the following chapters:

[Chapter 18: Installing ODBC Components](#), which describes how to install and remove ODBC components.

[Chapter 19: Configuring Data Sources](#), which describes how to modify information about data sources is stored in the system registry. This chapter applies to Windows users only.

Chapter 18: Installing ODBC Components

This chapter describes how ODBC components are installed and removed. Because driver developers always install an ODBC component (their driver), they need to read this chapter. Application developers need to read this chapter only if they will ship ODBC components with their application. ODBC components are defined to be the Driver Manager, drivers, translators, the installer DLL, the cursor library, and any related files. For the purposes of this chapter, ODBC applications are not considered to be ODBC components.

Note: This chapter is specific to Windows platforms. How ODBC components are installed on other platforms is platform-specific.

ODBC components are installed and removed on a component-by-component basis, not a file-by-file basis. For example, if a translator consists of the translator itself and a number of data files, these files are installed and removed as a group; they must not be installed and removed on a file-by-file basis. The reason for this is to make sure that only complete components exist on the system.

For purposes of installing and removing components, the following are defined to be the ODBC components:

- **Core components.** The Driver Manager, cursor library, installer DLL, and any other related files make up the core components and must be installed and removed as a group.
- **Drivers.** Each driver is a separate component.
- **Translators.** Each translator is a separate component.

Setup Program

The user runs the setup program to start the setup process. The setup program is written by the application or driver developer. In addition to installing ODBC components, it can install other software. For example, application developers might use the same setup program to install both ODBC components and their application.

Developers can write the setup program from scratch, using the Windows SDK setup utilities or setup software from other vendors. This gives them complete control over the setup program's look and feel. The setup program can be written to install additional software, such as an ODBC application. For more information on the Windows SDK setup utilities, see the Windows SDK documentation.

How much of the installation is actually done by the setup program depends on what functions it calls in the installer DLL. The installer DLL contains functions to install ODBC components in the following ways:

- **Install individual ODBC components.** The setup program calls **SQLInstallDriverManager**, **SQLInstallDriverEx**, or **SQLInstallTranslatorEx** in the installer DLL to retrieve the path of the directory in which the component is to be installed and to add information about the component to the registry. Note that these functions do not actually copy files; the setup program does this using the information in the arguments of these functions.

The installer DLL contains functions to remove ODBC components in the following ways:

Remove individual ODBC components. The setup program calls **SQLRemoveDriverManager**, **SQLRemoveDriver**, or **SQLRemoveTranslator** in the installer DLL to decrement a component's usage count in the registry and, if the component's new usage count falls to 0, remove all information about the component from the registry. Note that these functions do not actually remove the files for the component; the setup program does this if the new usage count falls to 0.

Installer DLL

The installer DLL contains functions to maintain registry information about data sources, install and remove ODBC components, and maintain registry information about those components. It is written by Microsoft and can be redistributed by users of the Microsoft Data Access SDK. For a complete description of the functions in the installer DLL, see “[Chapter 23: Installer DLL API Reference](#).”

Driver Setup DLL

The driver setup DLL contains the **ConfigDriver** and **ConfigDSN** functions. **ConfigDriver** performs driver-specific installation tasks, such as entering driver-specific information into the registry. **ConfigDSN** maintains driver-specific information about data sources in the registry. For a complete description of these functions, see “[Chapter 22: Setup DLL API Reference](#).”

The driver setup DLL is written by the driver developer. It can be part of the driver DLL or a separate DLL.

Usage Counting

The types of usage counts are maintained in the registry for each component: a component usage count and one or more optional file usage counts. The component usage count helps the installer DLL maintain registry entries. It is stored in the UsageCount value under the ODBC Core, driver, and translator subkeys. For the format of the UsageCount value and more information about these subkeys, see “[Registry Entries for ODBC Components](#)” later in this chapter.

When a component is first installed, the installer DLL creates a subkey for it and sets the data for the UsageCount value in that subkey to 1. When the component is installed again, the installer DLL increments the usage count. When the component is removed, the installer DLL decrements the usage count. If the usage count falls to 0, the installer DLL removes the subkey for the component.

Caution: An application should not physically remove Driver Manager files when the component usage count and file usage count reach zero.

File usage counts help determine when a file must actually be copied or deleted as opposed to incrementing or decrementing the usage count. This is important because ODBC components-and therefore the files in ODBC components-are shared and can be installed or removed by a variety of applications. The application can physically delete driver and translator files if the component usage count and the file usage count reach zero. Driver Manager files should not, however, be physically deleted when both the component usage count and the file usage count have reached zero, because these files may be used by other applications that have not incremented the file usage count.

Note: File usage counts are optional in Windows NT and Windows 95.

File usage counts are maintained by the setup program after it call **SQLInstallDriverManager**, **SQLInstallDriverEx**, **SQLInstallTranslatorEx**, **SQLRemoveDriverManager**, **SQLRemoveDriver**, or **SQLRemoveTranslator**.

When a component is first installed, the setup program or installer DLL creates a value under the following key for each file in that component that is not already on the system:

```
HKEY_LOCAL_MACHINE
SOFTWARE
Microsoft
Windows
CurrentVersion
SharedDlls
```

It sets the data for those values to 1 and copies the file to the system. When the component is installed again, the setup program or installer DLL increments the usage counts. When the component is removed, the setup program or installer DLL decrements the usage counts. If any usage count falls to 0, the setup program or installer DLL removes the value for the file, and if the component is a driver or a translator, deletes the file. Driver Manager files should not be deleted.

The format of the file usage count value is:

| Name | Data type | Data |
|------------------|-----------|--------------|
| <i>full-path</i> | REG_DWORD | <i>count</i> |

For example, suppose a driver for Informix uses the INFRMX32.DLL and INFRMX32.HLP files, and that this driver has been installed twice. The values under the SharedDlls subkey for the Informix driver would be:

```
C:\WINDOWS\SYSTEM32\INFRMX32.DLL : REG_DWORD : 0x2
C:\WINDOWS\SYSTEM32\INFRMX32.DLL : REG_DWORD : 0x2
```

Redistributable Files

A number of files that are shipped with the ODBC component of the Microsoft Data Access SDK may be redistributed by application and driver developers. All developers who ship ODBC drivers must redistribute the following files:

| ODBC component | File | Core Component? |
|--------------------|--------------|-----------------|
| Administrator | ODBCAD32.EXE | No |
| Code page | 12520437.CPX | No |
| | 12520850.CPX | No |
| | MSCPXL32.DLL | No |
| Connection pooling | MTXDM.DLL | No |
| Driver Manager | DS16GT.DLL | Yes |
| | DS32GT.DLL | Yes |
| | ODBC16GT.DLL | Yes |
| | ODBC32.DLL | Yes |
| | ODBC32GT.DLL | Yes |
| | ODBCCP32.CPL | Yes |
| | ODBCCP32.DLL | Yes |
| | ODBCCR32.DLL | Yes |
| | ODBCINT.DLL | Yes |
| | ODBCTRAC.DLL | Yes |
| | MSDADC.DLL | |
| | MSDAENUM.DLL | |
| | MSDAER.DLL | |
| | MSDAERR.DLL | |
| | MSDAPS.DLL | |
| | MSDASQL.DLL | |
| | MSDASQLR.DLL | |
| | MSDATL.DLL | |
| | MSDATT.DLL | |
| Header files | ODBCINST.H | No |
| | SQL.H | No |
| | SQLEXT.H | No |
| | SQLTYPES.H | No |
| | SQLUCODE.H | No |
| Help | ODBC.CNT | No |
| | ODBC.HLP | No |
| | ODBCINST.CNT | Yes |
| | ODBCINST.HLP | Yes |
| | SDKGUIDE.CNT | No |
| | SDKGUIDE.HLP | No |
| Support files | MSVCRT40.DLL | No |

Registry Entries for ODBC Components

The installer DLL maintains information in the registry about each installed ODBC component. On computers running Microsoft® Windows NT® and Microsoft Windows® 95/98, this information is stored in subkeys under the following key in the registry:

```
HKEY_LOCAL_MACHINE
    SOFTWARE
        ODBC
            Odbcinst.ini
```

Because Odbcinst.ini is a subkey of the HKEY_LOCAL_MACHINE tree, the information about ODBC components is available to all users of the machine. For information about the registry, see the Platform SDK documentation and the release notes for the ODBC component of the MDAC SDK on the Microsoft Web site.

ODBC Core Subkey

The value under the ODBC Core subkey gives the usage count for the core components (Driver Manager, cursor library, installer DLL, and so on). The format of this value is:

| Name | Data type | Data |
|------------|-----------|--------------|
| UsageCount | REG_DWORD | <i>count</i> |

For example, suppose the ODBC Core components have been installed by the setup programs for three different applications and two different drivers. The value under the ODBC Core subkey would be:

UsageCount : REG_DWORD : 0x5

ODBC Drivers Subkey

The values under the ODBC Drivers subkey list the installed drivers. The format of these values is:

| Name | Data type | Data |
|---------------------------|-----------|-----------|
| <i>driver-description</i> | REG_SZ | Installed |

where *driver-description* is defined by the driver developer. It is usually the name of the DBMS associated with the driver.

For example, suppose drivers have been installed for formatted text files and SQL Server. The values under the ODBC Drivers subkey might be:

Text : REG_SZ : Installed

SQL Server : REG_SZ : Installed

Driver Specification Subkeys

Each driver listed in the ODBC Drivers subkey has a subkey of its own. This subkey has the same name as the corresponding value under the ODBC Drivers subkey. The values under this subkey list the full paths of the driver and driver setup DLLs, the values of the driver keywords returned by **SQLDrivers**, and the usage count. The formats of the values are:

| Name | Data type | Data |
|------------------|-----------|---|
| APILevel | REG_SZ | 0 1 2 |
| ConnectFunctions | REG_SZ | {Y N}{Y N}{Y N} |
| CreateDSN | REG_SZ | <i>driver-description</i> |
| Driver | REG_SZ | <i>driver-DLL-path</i> |
| DriverODBCVer | REG_SZ | <i>nn.nn</i> |
| FileExtns | REG_SZ | <i>*.file-extension1[,*.file-extension2]...</i> |
| FileUsage | REG_SZ | 0 1 2 |
| Setup | REG_SZ | <i>setup-DLL-path</i> |
| SQLLevel | REG_SZ | 0 1 2 |
| UsageCount | REG_DWORD | <i>count</i> |

where the use of each keyword is:

| Keyword | Usage |
|-------------------------|--|
| APILevel | A number indicating the ODBC interface conformance level supported by the driver: 0 = None 1 = Level 1 supported 2 = Level 2 supported This must be the same as the value returned for the SQL_ODBC_INTERFACE_CONFORMANCE option in SQLGetInfo . |
| CreateDSN | The name of one or more data sources to be created when the driver is installed. The system information must include one data source specification section for each data source listed with the CreateDSN keyword. These sections should not include the Driver keyword, since this is specified in the driver specification section, but must include enough information for the ConfigDSN function in the driver setup DLL to create a data source specification without displaying any dialog boxes. For the format of a data source specification section, see “ Data Source Specification Subkeys ” in Chapter 19, “Configuring Data Sources.” |
| ConnectFunctions | A three-character string indicating whether the driver supports SQLConnect , SQLDriverConnect , and SQLBrowseConnect . If the driver supports SQLConnect , the first character is “Y”; otherwise, it is “N”. If the driver supports SQLDriverConnect , the second character is “Y”; otherwise, it is “N”. If the driver supports SQLBrowseConnect , the third character is “Y”; otherwise, it is “N”. For example, if a driver supports SQLConnect and SQLDriverConnect , but not SQLBrowseConnect , the three-character string is “YYN”. |
| DriverODBCVer | A character string with the version of ODBC that the driver supports. The version is of the form <i>nn.nn</i> , where the first two digits are the major version |

| | |
|------------------|--|
| | <p>and the next two digits are the minor version. For the version of ODBC described in this manual, the driver must return “03.00”.</p> <p>This must be the same as the value returned for the SQL_DRIVER_ODBC_VER option in SQLGetInfo.</p> |
| FileExtns | <p>For file-based drivers, a comma-separated list of extensions of the files the driver can use. For example, a dBASE driver might specify *.dbf and a formatted text file driver might specify *.txt,*.csv. For an example of how an application might use this information, see the FileUsage keyword.</p> |
| FileUsage | <p>A number indicating how a file-based driver directly treats files in a data source.</p> <p>0 = The driver is not a file-based driver. For example, an ORACLE driver is a DBMS-based driver.</p> <p>1 = A file-based driver treats files in a data source as tables. For example, an Xbase driver treats each Xbase file as a table.</p> <p>2 = A file-based driver treats files in a data source as a catalog. For example, a Microsoft Access driver treats each Microsoft Access file as a complete database.</p> <p>An application might use this to determine how users will select data. For example, Xbase and Paradox users often think of data as stored in files, while ORACLE and Microsoft Access users generally think of data as stored in tables.</p> <p>When a user selects Open Data File from the File menu, an application could display the Windows File Open common dialog box. The list of file types would use the file extensions specified with the FileExtns keyword for drivers that specify a FileUsage value of 1 and “Y” as the second character of the value of the ConnectFunctions keyword. After the user selects a file, the application would call SQLDriverConnect with the DRIVER keyword, then execute a SELECT * FROM table-name statement.</p> <p>When the user selects Import Data from the File menu, an application could display a list of descriptions for drivers that specify a FileUsage value of 0 or 2 and “Y” as the second character of the value of the ConnectFunctions keyword. After the user selects a driver, the application would call SQLDriverConnect with the DRIVER keyword, then display a custom Select Table dialog box.</p> |
| SQLLevel | <p>A number indicating the SQL92 grammar supported by the driver:</p> <p>0 = SQL92 Entry</p> <p>1 = FIPS127-2 Transitional</p> <p>2 = SQL92 Intermediate</p> <p>3 = SQL92 Full</p> <p>This must be the same as the value returned for the SQL_SQL_CONFORMANCE option in SQLGetInfo.</p> |

For information about usage counts, see “[Usage Counting](#)” earlier in this chapter.

For example, suppose a driver for formatted text files has a driver DLL named TEXT.DLL, a separate driver setup DLL named TXTSETUP.DLL, and has been installed three times. If the driver supports the Level 1 API conformance level, supports the Minimum SQL grammar conformance level, treats files as tables, and can use files with the .txt and .csv extensions, the values under the Text subkey might be:

```
APILevel : REG_SZ : 1
ConnectFunctions : REG_SZ : YYN
Driver : REG_SZ : C:\WINDOWS\SYSTEM32\TEXT.DLL
DriverODBCVer : REG_SZ : 03.00.00
FileExtns : REG_SZ : *.txt,*.csv
FileUsage : REG_SZ : 1
Setup : REG_SZ : C:\WINDOWS\SYSTEM32\TXTSETUP.DLL
SQLLevel : REG_SZ : 0
UsageCount : REG_DWORD : 0x3
```

Default Driver Subkey

The Default subkey contains a single value that describes the driver used by the default data source. The format of this value is:

| Name | Data type | Data |
|--------|-----------|-----------------------------------|
| Driver | REG_SZ | <i>default-driver-description</i> |

where *default-driver-description* is the same as the name of the value under the ODBC Drivers subkey that describes the driver.

For example, if the default data source uses the SQL Server driver, the value under the Default subkey might be:

Driver : REG_SZ : SQL Server

Note The default driver contained in the Default subkey can refer to either a default user DSN or a default system DSN. If both a default user DSN and a default system DSN have been created, the default driver is determined by the DSN that was created last, so it might not be a valid entry for the DSN that was created first.

ODBC Translators Subkey

The values under the ODBC Translators subkey list the installed translators. The format of these values is:

| Name | Data type | Data |
|------------------------|------------------|------------------|
| <i>translator-desc</i> | REG_SZ | Installed |

where *translator-desc* is defined by the translator developer.

For example, suppose a user has installed the Microsoft Code Page Translator and a custom ASCII to EBCDIC translator. The values under the ODBC Translators subkey might be:

```
MS Code Page Translator: REG_SZ : Installed
ASCII to EBCDIC: REG_SZ : Installed.
```

Translator Specification Subkeys

Each translator listed in the ODBC Translators subkey has a subkey of its own. This subkey has the same name as the corresponding value under the ODBC Translators subkey. The values under this subkey list the full paths of the translator and translator setup DLLs and the usage count. The formats of the values are as shown in the following table.

| Name | Data type | Data |
|------------|-----------|----------------------------|
| Translator | REG_SZ | <i>translator-DLL-path</i> |
| Setup | REG_SZ | <i>setup-DLL-path</i> |
| UsageCount | REG_DWORD | <i>count</i> |

For example, suppose the Microsoft® Code Page Translator has a translation DLL named Mscpxl32.dll, that the translator setup functions are in the same DLL, and that the translator has been installed three times. The values under the Microsoft Code Page Translator subkey might be as follows:

Translator : REG_SZ : C:\WINDOWS\SYSTEM32\MSCPXL32.DLL
Setup : REG_SZ : C:\WINDOWS\SYSTEM32\MSCPXL32.DLL

UsageCount : REG_DWORD : 0x3

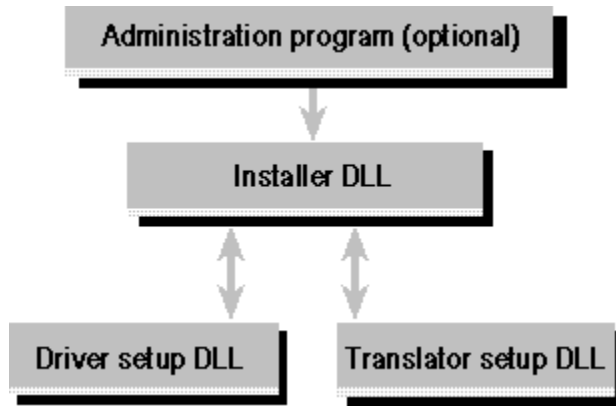
Chapter 19: Configuring Data Sources

Information about data sources is stored in the system registry. Users modify data source information through an administration program. This can be the ODBC Administrator shipped with the Microsoft Data Access (DA) SDK, the ODBC Control Panel device, or an administration program written by an application or driver developer.

Note: This chapter is specific to Windows platforms. How data sources are configured on other platforms is platform-specific.

Configuration Components

Data sources are configured by the installer DLL, which in turn calls driver setup DLLs and translator setup DLLs as needed. The installer DLL is either invoked directly from the Control Panel or loaded and called by another program, known as the *administration program*. The following figure shows the relationship between the configuration components.



Configuration components

Administration Program

An administration program, the ODBC Administrator, is shipped with the Data Access SDK and can be redistributed by users of the SDK. In addition, developers can write their own administration programs. Generally, developers write their own administration programs only if they want to retain complete control over data source configuration, or if they are configuring data sources directly from an application that is acting as an administration program. For example, a spreadsheet program might allow users to add and then use data sources at run time.

The administration program first loads the installer DLL. It then calls functions in the installer DLL to perform the following tasks:

- Add, modify, or delete data sources interactively. The administration program can call **SQLManageDataSources**, **SQLCreateDataSource**, or **SQLConfigDataSource**.

SQLManageDataSources displays a dialog box with which the user can add, modify, or delete data sources and specify tracing options; this function is called when the installer DLL is invoked directly from the Control Panel. **SQLCreateDataSource** displays a dialog box with which the user can only add data sources. **SQLConfigDataSource** passes the call directly to the driver setup DLL.

In all cases, the installer DLL calls **ConfigDSN** in the driver setup DLL to actually add, modify, or delete the data source. The driver setup DLL may prompt the user for additional information.

- Add, modify, or delete data sources silently. The administration program calls **SQLConfigDataSource** in the installer DLL and passes it a null window handle, the name of a data source to add, modify, or delete, and a list of values for the registry. The installer DLL calls **ConfigDSN** in the driver setup DLL to actually add, modify, or delete the data source.

Add, modify, or delete a default data source. The default data source is the same as any other data source, except that its name is Default. It is added, modified, or deleted in the same fashion as any other data source.

Installer DLL

The installer DLL contains functions to install and remove ODBC components, maintain registry information about those components, and maintain registry information about data sources. It is written by Microsoft and can be redistributed by users of the Microsoft Data Access SDK. For a complete description of the functions in the installer DLL, see “[Chapter 23: Installer DLL API Reference](#).”

Driver Setup DLLs

The driver setup DLL contains the **ConfigDriver** and **ConfigDSN** functions. **ConfigDSN** maintains driver-specific information about data sources in the registry. **ConfigDriver** performs driver-specific installation tasks, such as entering driver-specific information into the registry. For a complete description of these functions, see “[Chapter 22: Setup DLL API Reference](#).”

ConfigDSN calls the following functions in the installer DLL to maintain data source information in the registry:

- **SQLWriteDSNToIni.** Add a data source.
- **SQLRemoveDSNFromIni.** Delete a data source.
- **SQLWritePrivateProfileString.** Write a driver-specific value under a data source specification subkey.
- **SQLGetPrivateProfileString.** Read a driver-specific value from a data source specification subkey.
- **SQLGetTranslator.** Prompt the user for a translator name and option. This function calls **ConfigTranslator** in the translator setup DLL.

The driver setup DLL is written by the driver developer. It can be part of the driver DLL or a separate DLL.

Translator Setup DLLs

The translator setup DLL contains the **ConfigTranslator** function, which returns the default option for a translator. If necessary, it prompts the user for this information. For a complete description of this function, see “[Chapter 22: Setup DLL API Reference](#).”

The translator setup DLL is written by the translator developer. It can be part of the translator DLL or a separate DLL.

Registry Entries for Data Sources

The installer DLL maintains information in the registry about each data source. On Windows NT and Windows 95, this information is stored in subkeys under one of the following two keys in the registry:

HKEY_LOCAL_MACHINE
SOFTWARE
ODBC
ODBC.INI

HKEY_CURRENT_USER
SOFTWARE
ODBC
ODBC.INI

Which key is used depends on whether the data source is a *system data source*, which is available to all users, or a *user data source*, which is available only to the current user. System data sources are stored on the HKEY_LOCAL_MACHINE tree and user data sources are stored on the HKEY_CURRENT_USER tree. In all other respects, system data sources and user data sources are identical. For information on the registry, see the Win32 SDK documentation.

ODBC Data Sources Subkey

The values under the ODBC Data Sources subkey list the data sources. The format of these values is:

| Name | Data type | Data |
|-------------------------|------------------|---------------------------|
| <i>data-source-name</i> | REG_SZ | <i>driver-description</i> |

where data-source-name is defined by the administration program (which usually prompts the user for it) and driver-description is defined by the driver developer (it is usually the name of the DBMS associated with the driver).

For example, suppose three data sources have been defined: Inventory, which uses SQL Server; Payroll, which uses dBase; and Personnel, which uses formatted text files. The values under the ODBC Data Sources subkey might be:

```
Inventory   : REG_SZ   : SQL Server
Payroll     : REG_SZ   : dBase
Personnel   : REG_SZ   : Text
```

Data Source Specification Subkeys

Each data source listed in the ODBC Data Sources subkey has a subkey of its own. This subkey has the same name as the corresponding value under the ODBC Data Sources subkey. The values under this subkey must list the driver DLL and may list a description of the data source. If the driver supports translators, the values may list the name of a default translator, the default translation DLL, and the default translation option. The values may also list other information required by the driver to connect to the data source. For example, the driver might require a server name, database name, or schema name.

The formats of the values are as follows. Only the Driver value is required.

| Name | Data type | Data |
|-----------------------|-----------------------|----------------------------|
| Description | REG_SZ | <i>description</i> |
| Driver | REG_SZ | <i>driver-DLL-path</i> |
| TranslationDLL | REG_SZ | <i>translator-DLL-path</i> |
| TranslationName | REG_SZ | <i>translator-name</i> |
| TranslationOption | REG_SZ | <i>translation-option</i> |
| <i>opt-value-name</i> | <i>opt-value-type</i> | <i>opt-value-data</i> |

For example, suppose the SQL Server driver requires the server name and a flag for OEM to ANSI conversion, and defines the Server and OEMTOANSI values for these. Suppose also that the Inventory data source uses the Microsoft Code Page Translator to translate between the Windows Latin 1 (1250) and Multilingual (850) code pages. The values under the Inventory subkey might be:

```
Description : REG_SZ : Inventory database on server InvServ
Driver : REG_SZ : C:\WINDOWS\SYSTEM32\SQLSRV32.DLL
OEMTOANSI : REG_SZ : Yes
Server : REG_SZ : InvServ
TranslationDLL : REG_SZ : C:\WINDOWS\SYSTEM32\MSCPXL32.DLL
TranslationName : REG_SZ : MS Code Page Translator
TranslationOption : REG_SZ : 12500850
```

Default Subkey

The registry may specify a default data source with the Default subkey. This subkey is a special case of a data source specification subkey and has the same values as any other data source specification subkey. The only difference is that it is not listed as a value under the ODBC Data Sources subkey.

ODBC Subkey

The values under the ODBC subkey specify ODBC tracing options. These options are set through the Tracing tab of the ODBC Data Source Administrator dialog box displayed by **SQLManageDataSources**. The ODBC subkey itself is optional. The format of these values is:

| Name | Data type | Data |
|---------------|-----------|-----------------------|
| Trace | REG_SZ | 0 1 |
| TraceAutoStop | REG_SZ | 0 1 |
| TraceFile | REG_SZ | <i>tracefile-path</i> |

where the values have the following meanings:

| Value | Meaning |
|---------------|---|
| Trace | <p>If the Trace value is set to 1 when an application calls SQLAllocHandle with the SQL_HANDLE_ENV option, then tracing is enabled for the calling application.</p> <p>If the Trace keyword is set to 0 when an application calls SQLAllocHandle with the SQL_HANDLE_ENV option, then tracing is disabled for the calling application. This is the default value.</p> <p>An application can enable or disable tracing with the SQL_ATTR_TRACE connection attribute. However, doing so does not change the data for this value.</p> |
| TraceFile | <p>If tracing is enabled, the Driver Manager writes to the trace file specified by the TraceFile value.</p> <p>If no trace file is specified, the Driver Manager writes to the \SQL.LOG file on the current drive. This is the default value.</p> <p>Tracing should only be used for a single application or each application should specify a different trace file. Otherwise, two or more applications will attempt to open the same trace file at the same time, causing an error.</p> <p>An application can specify a new trace file with the SQL_ATTR_TRACEFILE connection attribute. However, doing so does not change the data for this value.</p> |
| TraceAutoStop | <p>If the TraceAutoStop value is set to 1 when an application calls SQLFreeHandle with the SQL_HANDLE_ENV option, then tracing is disabled for all applications and the Trace value is set to 0. This is the default value.</p> <p>If the TraceAutoStop value is set to 0, then tracing must be disabled through the Tracing tab of the ODBC Data Source Administrator dialog box displayed by the SQLManageDataSources function.</p> |

For example, suppose that tracing is enabled, the trace file is C:\ODBC.LOG, and that tracing is to stop automatically. The values under the ODBC subkey would be:

```
Trace : REG_SZ : 1
TraceAutoStop : REG_SZ : 1
TraceFile : REG_SZ : C:\ODBC.LOG
```


Section 2: DLL Functions

Section 2 of this *ODBC Programmer's Reference* excerpt contains the following chapters:

[Chapter 21: Installer DLL API Function Summary](#), which describes how to install and remove ODBC components.

[Chapter 22: Setup DLL API Reference](#), which describes describes the syntax of the driver setup DLL API.

[Chapter 23: Installer DLL API Reference](#), which describes the syntax of the functions in the installer DLL API.

[Chapter 24: Translation DLL Function Reference](#), describes the syntax of the translation DLL API

Chapter 21: Installer DLL API Function Summary

The following table describes the functions in the installer DLL. For more information about the syntax and semantics for each function, see “Chapter 23: Installer DLL API Reference.”

| Task | Function name | Purpose |
|--------------------------|--|---|
| Installing ODBC | SQLConfigDriver | Loads the driver-specific setup DLL. |
| | SQLGetInstalledDrivers | Returns a list of installed drivers. |
| | SQLInstallDriverEx | Adds a driver to the system information. |
| | SQLInstallDriverManager | Returns the target directory for the Driver Manager. |
| | SQLInstallerError | Returns error or status information for the installer functions. |
| | SQLInstallTranslatorEx | Adds a translator to the system information. |
| | SQLPostInstallerError | Allows a driver or translator setup library to report errors. |
| | SQLRemoveDriver | Removes a driver from the system information. |
| | SQLRemoveDriverManager | Removes ODBC core components from the system information. |
| | SQLRemoveTranslator | Removes the translator from the system information. |
| | SQLConfigDataSource | Calls the driver-specific setup DLL. |
| Configuring data sources | SQLCreateDataSource | Displays a dialog box to add a data source. |
| | SQLGetConfigMode | Retrieves the configuration mode that indicates where the ODBC.ini entry listing DSN values is in the system information. |
| | SQLGetPrivateProfileString | Writes a value to the system information. |
| | SQLGetTranslator | Displays a dialog box to select a translator. |
| | SQLManageDataSources | Displays a dialog box to configure data sources and drivers. |
| | SQLReadFileDSN | Reads information from file DSNs. |
| | SQLRemoveDefaultDataSource | Removes the default data source. |
| | SQLRemoveDSNFromIni | Removes a data source. |
| | SQLGetConfigMode | Sets the configuration mode that indicates where the ODBC.ini entry listing DSN values is in the system information. |
| | SQLValidDSN | Checks the length and validity of the data source name. |
| | SQLWriteDSNTToIni | Adds a data source. |
| | SQLWriteFileDSN | Writes information to file DSNs. |
| | SQLWritePrivateProfileString | Gets a value from the system information. |

Chapter 22: Setup DLL API Reference

This chapter describes the syntax of the driver setup DLL API, which consists of two functions (**ConfigDriver** and **ConfigDSN**). **ConfigDriver** and **ConfigDSN** may be either in the driver DLL or in a separate setup DLL.

In addition, this chapter describes the syntax of the translator setup DLL API, which consists of a single function (**ConfigTranslator**). **ConfigTranslator** may be either in the translator DLL or in a separate setup DLL.

Each function is labeled with the version of ODBC in which it was introduced.

ConfigDriver

Conformance

Version Introduced: ODBC 2.5

Summary

ConfigDriver allows a setup program to perform install and uninstall functions without requiring the program to call **ConfigDSN**. This function will perform driver-specific functions such as creating driver-specific system information and performing DSN conversions during installation, and cleaning up system information modifications during uninstall. This function is exposed by the driver setup DLL or a separate setup DLL.

Syntax

```
BOOL ConfigDriver (  
    HWND  hwndParent,  
    WORD  fRequest,  
    LPCSTR lpszDriver,  
    LPCSTR lpszArgs,  
    LPSTR  lpszMsg,  
    WORD  cbMsgMax,  
    WORD * pcbMsgOut);
```

Arguments

hwndParent

[Input]

Parent window handle. The function will not display any dialog boxes if the handle is null.

fRequest

[Input]

Type of request. *fRequest* must contain one of the following values:

ODBC_INSTALL_DRIVER: Install a new driver.

ODBC_REMOVE_DRIVER: Remove a driver.

This option can also be driver-specific, in which case the *fRequest* for the first option must start from ODBC_CONFIG_DRIVER_MAX+1. The *fRequest* for any additional option must also start from a value greater than ODBC_CONFIG_DRIVER_MAX+1.

lpszDriver

[Input]

The name of the driver as registered in the ODBCINST.INI key of the system information.

lpszArgs

[Input]

A null-terminated string containing arguments for a driver-specific *fRequest*.

lpszMsg

[Output]

A null-terminated string containing an output message from the driver setup.

cbMsgMax

[Input]

Length of *lpszMsg*.

pcbMsgOut

[Output]

Total number of bytes available to return in *lpszMsg*.

If the number of bytes available to return is greater than or equal to *cbMsgMax*, the output message in *lpszMsg* is truncated to *cbMsgMax* minus the null-termination character. The *pcbMsgOut* argument can be a null pointer.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **ConfigDriver** returns FALSE, an associated **pfErrorCode* value is posted to the installer error buffer by a call to **SQLPostInstallerError**, and may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------------|--------------------------------------|---|
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid. |
| ODBC_ERROR_INVALID_REQUEST | Invalid type of request | The <i>fRequest</i> argument was not one of the following: ODBC_INSTALL_DRIVER. ODBC_REMOVE_DRIVER. The driver-specific option was less than or equal to ODBC_CONFIG_DRIVER_MAX. |
| ODBC_ERROR_INVALID_DRIVER_NAME | Invalid driver or translator name. | The <i>lpszDriver</i> argument was invalid. It could not be found in the registry. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | Could not perform the operation requested by the <i>fRequest</i> argument. |
| ODBC_ERROR_DRIVER_SPECIFIC | Driver- or translator-specific error | A driver-specific error for which there is no defined ODBC installer error. The <i>SzError</i> argument in a call to the SQLPostInstallerError function should contain the driver-specific error message |

Comments

Driver-Specific Options

An application can request driver-specific features exposed by the driver by using the *fRequest* argument. The *fRequest* for the first option will be ODBC_CONFIG_DRIVER_MAX plus 1, and additional options will be incremented by 1 from that value. Any arguments required by the driver for that function should be provided in a null-terminated string passed in the *lpszArgs* argument. Drivers providing such functionality should maintain a table of driver-specific options. The options should be fully documented in driver documentation. Application writers who use driver-specific options should be aware that this will make the application less interoperable.

Messages

A driver setup routine can send a text message to an application as a null-terminated string in the *lpszMsg* buffer. The message will be truncated to *cbMsgMax* minus the null-termination character by the **ConfigDriver** function if it is greater than or equal to *cbMsgMax* characters.

ConfigDSN

Conformance

Version Introduced: ODBC 1.0

Summary

ConfigDSN adds, modifies, or deletes data sources from the system information. It may prompt the user for connection information. It can be in the driver DLL or a separate setup DLL.

Syntax

```
BOOL ConfigDSN(  
    HWND hwndParent,  
    WORD fRequest,  
    LPCSTR lpszDriver,  
    LPCSTR lpszAttributes);
```

Arguments

hwndParent

[Input]

Parent window handle. The function will not display any dialog boxes if the handle is null.

fRequest

[Input]

Type of request. *fRequest* must contain one of the following values:

ODBC_ADD_DSN: Add a new data source.

ODBC_CONFIG_DSN: Configure (modify) an existing data source.

ODBC_REMOVE_DSN: Remove an existing data source.

lpszDriver

[Input]

Driver description (usually the name of the associated DBMS) presented to users instead of the physical driver name.

lpszAttributes

[Input]

List of attributes in the form of keyword-value pairs. For information about the list structure, see “Comments.”

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **ConfigDSN** returns FALSE, an associated **pfErrorCode* value is posted to the installer error buffer by a call to **SQLPostInstallerError**, and may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|----------------------------------|--------------------------------------|---|
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid. |
| ODBC_ERROR_INVALID_KEYWORD_VALUE | Invalid keyword-value pairs | The <i>lpszAttributes</i> argument contained a syntax error. |
| ODBC_ERROR_INVALID_D_NAME | Invalid driver or translator name. | The <i>lpszDriver</i> argument was invalid. It could not be found in the registry. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request | The <i>fRequest</i> argument was not one of the following. ODBC_ADD_DSN ODBC_CONFIG_DSN ODBC_REMOVE_DSN |
| ODBC_ERROR_DRIVER_SPECIFIC | Driver- or translator-specific error | A driver-specific error for which there is no defined ODBC installer error. The <i>SzError</i> argument in a call to the SQLPostInstallerError function should contain the driver-specific error message |
| ODBC_ERROR_INVALID_OPTION | Invalid translation option | The <i>pvOption</i> argument contained an invalid value. |

Comments

ConfigDSN receives connection information from the installer DLL as a list of attributes in the form of keyword-value pairs. Each pair is terminated with a null byte and the entire list is terminated with a null byte (that is, two null bytes mark the end of the list). Spaces are not allowed around the equal sign in the keyword-value pair. **ConfigDSN** can accept keywords that are not valid keywords for **SQLBrowseConnect** and **SQLDriverConnect**. **ConfigDSN** does not necessarily support all keywords that are valid keywords for **SQLBrowseConnect** and **SQLDriverConnect**. (**ConfigDSN** does not accept the DRIVER keyword.) The keywords used by the **ConfigDSN** function must support all the options required to re-create the data source using the AUTO setup feature of the installer. When the uses of the **ConfigDSN** values and the connection string values are the same, the same keywords should be used.

As in **SQLBrowseConnect** and **SQLDriverConnect**, the keywords and their values should not contain the []{}(),;?*=!@ characters, and the value of the DSN keyword cannot consist only of blanks. Because of the registry grammar, keywords and data source names cannot contain the backslash (\) character.

ConfigDSN should call **SQLValidDSN** to check the length of the data source name, and to verify that no invalid characters are included in the name. If the data source name is longer than **SQL_MAX_DSN_LENGTH** or includes invalid characters, then **SQLValidDSN** returns an error, and **ConfigDSN** returns an error. The length of the data source name is also checked by **SQLWriteDSNToIni**.

For example, to configure a data source that requires a user ID, password, and database name, a setup application might pass the following keyword-value pairs:

DSN=Personnel Data\0UID=Smith\0PWD=Sesame\0DATABASE=Personnel\0\0

For more information about these keywords, see **SQLDriverConnect** and each driver's documentation.

To display a dialog box, *hwndParent* must not be null.

Adding a Data Source

If a data source name is passed to **ConfigDSN** in *lpszAttributes*, **ConfigDSN** checks that the name is valid. If the data source name matches an existing data source name and *hwndParent* is null, **ConfigDSN** overwrites the existing name. If it matches an existing name and *hwndParent* is not null, **ConfigDSN** prompts the user to overwrite the existing name.

If *lpszAttributes* contains enough information to connect to a data source, **ConfigDSN** can add the data source or display a dialog box with which the user can change the connection information. If *lpszAttributes* does not contain enough information to connect to a data source, **ConfigDSN** must determine the necessary information; if *hwndParent* is not null, it displays a dialog box to retrieve the information from the user.

If **ConfigDSN** displays a dialog box, it must display any connection information passed to it in *lpszAttributes*. In particular, if a data source name was passed to it, **ConfigDSN** displays that name but does not allow the user to change it. **ConfigDSN** can supply default values for connection information not passed to it in *lpszAttributes*.

If **ConfigDSN** cannot get complete connection information for a data source, it returns FALSE.

If **ConfigDSN** can get complete connection information for a data source, it calls **SQLWriteDSNToIni** in the installer DLL to add the new data source specification to the ODBC.INI file (or registry). **SQLWriteDSNToIni** adds the data source name to the [ODBC Data Sources] section, creates the data source specification section, and adds the DRIVER keyword with the driver description as its value. **ConfigDSN** calls **SQLWritePrivateProfileString** in the installer DLL to add any additional keywords and values used by the driver.

Modifying a Data Source

To modify a data source, a data source name must be passed to **ConfigDSN** in *lpszAttributes*. **ConfigDSN** checks that the data source name is in the ODBC.INI file (or registry).

If *hwndParent* is null, **ConfigDSN** uses the information in *lpszAttributes* to modify the information in the ODBC.INI file (or registry). If *hwndParent* is not null, **ConfigDSN** displays a dialog box using the information in *lpszAttributes*; for information not in *lpszAttributes*, it uses information from the system information. The user can modify the information before **ConfigDSN** stores it in the system information.

If the data source name was changed, **ConfigDSN** first calls **SQLRemoveDSNFromIni** in the installer DLL to remove the existing data source specification from the ODBC.INI file (or registry). It then follows the steps in the previous section to add the new data source specification. If the data source name was not changed, **ConfigDSN** calls **SQLWritePrivateProfileString** in the installer DLL to make any other changes. **ConfigDSN** may not delete or change the value of the Driver keyword.

Deleting a Data Source

To delete a data source, a data source name must be passed to **ConfigDSN** in *lpszAttributes*. **ConfigDSN** checks that the data source name is in the ODBC.INI file (or registry). It then calls **SQLRemoveDSNFromIni** in the installer DLL to remove the data source.

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a data source | SQLConfigDataSource |
| Getting a value from the ODBC.INI file or the registry | SQLGetPrivateProfileString |
| Removing the default data source | SQLRemoveDefaultDataSource |
| Removing a data source name from ODBC.INI (or registry) | SQLRemoveDSNFromIni |
| Adding a data source name to ODBC.INI (or registry) | SQLWriteDSNToIni |
| Writing a value to the ODBC.INI file or the registry | SQLWritePrivateProfileString |

ConfigTranslator

Conformance

Version Introduced: ODBC 2.0

Summary

ConfigTranslator returns a default translation option for a translator. It can be in the translator DLL or a separate setup DLL.

Syntax

```
BOOL ConfigTranslator(  
    HWND hwndParent,  
    DWORD * pvOption);
```

Arguments

hwndParent

[Input]Parent window handle. The function will not display any dialog boxes if the handle is null.

pvOption

[Output]

A 32-bit translation option.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **ConfigTranslator** returns FALSE, an associated **pfErrorCode* value is posted to the installer error buffer by a call to **SQLPostInstallerError**, and may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|----------------------------|--------------------------------------|---|
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid or NULL. |
| ODBC_ERROR_DRIVER_SPECIFIC | Driver- or translator-specific error | A driver-specific error for which there is no defined ODBC installer error. The <i>SzError</i> argument in a call to the SQLPostInstallerError function should contain the driver-specific error message |
| ODBC_ERROR_INVALID_OPTION | Invalid translation option | The <i>pvOption</i> argument contained an invalid value. |

Comments

If the translator supports only a single translation option, **ConfigTranslator** returns TRUE and sets *pvOption* to the 32-bit option. Otherwise, it determines the default translation option to use. **ConfigTranslator** can display a dialog box with which a user selects a default translation option.

Related Functions

| For information about | See |
|------------------------------|---|
| Getting a translation option | SQLGetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |
| Selecting a translator | SQLGetTranslator |
| Setting a translation option | SQLSetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |

Chapter 23: Installer DLL API Reference

This chapter describes the syntax of the functions in the installer DLL API. The installer DLL API consists of twenty functions. Three of these functions, **SQLGetTranslator**, **SQLRemoveDSNFromIni**, and **SQLWriteDSNToIni**, are called only by setup DLLs. The other functions are called by the setup and administration programs.

Each function is labeled with the version of ODBC in which it was introduced.

SQLConfigDataSource

Conformance

Version Introduced: ODBC 1.0

Summary

SQLConfigDataSource adds, modifies, or deletes data sources.

Syntax

```
BOOL SQLConfigDataSource(  
    HWND   hwndParent,  
    WORD   fRequest,  
    LPCSTR  lpzDriver,  
    LPCSTR  lpzAttributes);
```

Arguments

hwndParent

[Input]

Parent window handle. The function will not display any dialog boxes if the handle is null.

fRequest

[Input]

Type of request. *fRequest* must contain one of the following values:

ODBC_ADD_DSN: Add a new user data source.

ODBC_CONFIG_DSN: Configure (modify) an existing user data source.

ODBC_REMOVE_DSN: Remove an existing user data source.

ODBC_ADD_SYS_DSN: Add a new system data source.

ODBC_CONFIG_SYS_DSN: Modify an existing system data source.

ODBC_REMOVE_SYS_DSN: Remove an existing system data source.

ODBC_REMOVE_DEFAULT_DSN: Remove the default data source specification section from the system information. It also removes the default driver specification section from the ODBCINST.INI entry in the system information. (This *fRequest* performs the same function as the deprecated

SQLRemoveDefaultDataSource function.) When this option is specified, all of the other parameters in the call to **SQLConfigDataSource** should be NULLs; if they are not NULL, they will be ignored.

lpzDriver

[Input]

Driver description (usually the name of the associated DBMS) presented to users instead of the physical driver name.

lpzAttributes

[Input]

List of attributes in the form of keyword-value pairs. For more information, see **ConfigDSN** in “[Chapter 22: Setup DLL API Reference](#).”

Returns

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE.

Diagnostics

When **SQLConfigDataSource** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|----------------------------------|--|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request. | The <i>fRequest</i> argument was not one of the following: ODBC_ADD_DSN ODBC_CONFIG_DSN ODBC_REMOVE_DSN ODBC_ADD_SYS_DSN ODBC_CONFIG_SYS_DSN ODBC_REMOVE_SYS_DSN ODBC_REMOVE_DEFAULT_DSN |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The <i>fRequest</i> argument was not one of the following: ODBC_INSTALL_DRIVER ODBC_REMOVE_DRIVER The <i>fRequest</i> argument was a driver-specific option that was less than or equal to ODBC_CONFIG_DRIVER_MAX. |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszDriver</i> argument was invalid. It could not be found in the registry. |
| ODBC_ERROR_INVALID_KEYWORD_VALUE | Invalid keyword-value pairs | The <i>lpszAttributes</i> argument contained a syntax error. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The installer could not perform the operation requested by the <i>fRequest</i> argument. The call to ConfigDriver failed. |
| ODBC_ERROR_LOAD_LIBRARY_FAILED | Could not load the driver or translator setup library. | The driver setup library could not be loaded. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLConfigDataSource uses the value of *lpzDriver* to read the full path of the setup DLL for the driver from the system information. It loads the DLL and calls **ConfigDSN** with the same arguments that were passed to it.

SQLConfigDataSource returns FALSE if it is unable to find or load the setup DLL, or if the user cancels the dialog box. Otherwise, it returns the status it received from **ConfigDSN**.

SQLConfigDataSource maps the System DSN *fRequests* to the User DSN *fRequests* (ODBC_ADD_SYS_DSN to ODBC_ADD_DSN, ODBC_CONFIG_SYS_DSN to ODBC_CONFIG_DSN, and ODBC_REMOVE_SYS_DSN to ODBC_REMOVE_DSN). To distinguish user and System DSNs, **SQLConfigDataSource** sets the installer configuration mode according to the following table. Prior to returning, **SQLConfigDataSource** resets configuration mode to BOTHDSN.

| fRequest | Configuration mode |
|---------------------|---------------------------|
| ODBC_ADD_DSN | USERDSN_ONLY |
| ODBC_CONFIG_DSN | USERDSN_ONLY |
| ODBC_REMOVE_DSN | USERDSN_ONLY |
| ODBC_ADD_SYS_DSN | SYSTEMDSN_ONLY |
| ODBC_CONFIG_SYS_DSN | SYSTEMDSN_ONLY |
| ODBC_REMOVE_SYS_DSN | SYSTEMDSN_ONLY |

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a data source | ConfigDSN (in the setup DLL) |
| Removing a data source name from the system information | SQLRemoveDSNFromIni |
| Adding a data source name to the system information | SQLWriteDSNToIni |

SQLConfigDriver

Conformance

Version Introduced: ODBC 2.5

Summary

SQLConfigDriver loads the appropriate driver setup DLL and calls the **ConfigDriver** function.

Syntax

```
BOOL SQLConfigDriver (  
    HWND  hwndParent,  
    WORD  fRequest,  
    LPCSTR lpszDriver,  
    LPCSTR lpszArgs,  
    LPSTR  lpszMsg,  
    WORD  cbMsgMax,  
    WORD * pcbMsgOut);
```

Arguments

hwndParent

[Input]

Parent window handle. The function will not display any dialog boxes if the handle is null.

fRequest

[Input]

Type of request. *fRequest* must contain one of the following values:

ODBC_CONFIG_DRIVER: Changes the connection pooling timeout used by the driver.

ODBC_INSTALL_DRIVER: Installs a new driver.

ODBC_REMOVE_DRIVER: Removes an existing driver.

This option can also be driver-specific, in which case the *fRequest* for the first option must start from ODBC_CONFIG_DRIVER_MAX+1. The *fRequest* for any additional option must also start from a value greater than ODBC_CONFIG_DRIVER_MAX+1.

lpszDriver

[Input]

The name of the driver as registered in the system information.

lpszArgs

[Input]

A null-terminated string containing arguments for a driver-specific *fRequest*.

lpszMsg

[Output]

A null-terminated string containing an output message from the driver setup.

cbMsgMax

[Input]

Length of *lpszMsg*.

pcbMsgOut

[Output]

Total number of bytes available to return in *lpszMsg*. If the number of bytes available to return is greater than or equal to *cbMsgMax*, the output message in *lpszMsg* is truncated to *cbMsgMax* minus the null-termination character. The *pcbMsgOut* argument can be a null pointer.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLConfigDriver** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|---------------------------------|-----------------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFFER | Invalid buffer length | The <i>lpszMsg</i> argument was invalid. |
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request. | The <i>lpszMsg</i> argument was invalid. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The <i>fRequest</i> argument was not one of the following: ODBC_INSTALL_DRIVER ODBC_REMOVE_DRIVER The <i>fRequest</i> argument was a driver-specific option that was less than or equal to ODBC_CONFIG_DRIVER_MAX. |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszDriver</i> argument was invalid. It could not be found in the registry. |
| ODBC_ERROR_INVALID_KEYWORD_ | Invalid keyword-value pairs | The <i>lpszAttributes</i> argument contained a syntax error. |

| | | |
|------------------------------------|--|--|
| VALUE | | |
| ODBC_ERROR_REQUE ST_FAILED | Request failed | The installer could not perform the operation requested by the <i>fRequest</i> argument. The call to ConfigDriver failed. |
| ODBC_ERROR_LOAD_ LIBRARY_FAILED | Could not load the driver or translator setup library. | The driver setup library could not be loaded. |
| ODBC_ERROR_ OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLConfigDriver allows an application to call a driver's **ConfigDriver** routine without having to know the name and load the driver-specific setup DLL. A setup program calls this function after the driver setup DLL has been installed. The calling program should be aware that this function may not be available for all drivers. In such a case, the calling program should continue without error.

Driver-Specific Options

An application can request driver-specific features exposed by the driver by using the *fRequest* argument. The *fRequest* for the first option will be ODBC_CONFIG_DRIVER_MAX+1, and additional options will be incremented by 1 from that value. Any arguments required by the driver for that function should be provided in a null-terminated string passed in the *lpszArgs* argument. Drivers providing such functionality should maintain a table of driver-specific options. The options should be fully documented in driver documentation. Application writers who make use of driver-specific options should be aware that this use will make the application less interoperable.

Setting Connection Pooling Timeout

Connection pooling timeout properties can be set when setting the configuration of the driver.

SQLConfigDriver is called with an *fRequest* of ODBC_CONFIG_DRIVER and *lpszArgs* set to CPTimeout. CPTimeout determines the amount of time that a connection can remain in the connection pool without being used. When the timeout expires, the connection is closed and removed from the pool. The default timeout is 60 seconds.

When **SQLConfigDriver** is called with *fRequest* set to ODBC_INSTALL_DRIVER or ODBC_REMOVE_DRIVER, the Driver Manager loads the appropriate driver setup DLL and calls the **ConfigDriver** function. When **SQLConfigDriver** is called with an *fRequest* of ODBC_CONFIG_DRIVER, all processing is performed in the ODBC installer, so the driver setup DLL does not need to be loaded.

Messages

A driver setup routine can send a text message to an application as null-terminated strings in the *lpszMsg* buffer. The message will be truncated to *cbMsgMax* minus the null-termination character by the **ConfigDriver** function if it is greater than or equal to *cbMsgMax* characters.

Related Functions

| For information about | See |
|---|---|
| Adding, modifying, or removing a driver | ConfigDriver (in the setup DLL) |
| Removing the default data source | SQLRemoveDefaultDataSource |

SQLCreateDataSource

Conformance

Version Introduced: ODBC 2.0

Summary

SQLCreateDataSource displays a dialog box with which the user can add a data source.

Syntax

```
BOOL SQLCreateDataSource(  
    HWND hwnd,  
    LPSTR lpszDS);
```

Arguments

hwnd

[Input]
Parent window handle.

lpszDS

[Input]
Data source name. *lpszDS* can be a null pointer or an empty string.

Returns

SQLCreateDataSource returns TRUE if the data source is created. Otherwise, it returns FALSE.

Diagnostics

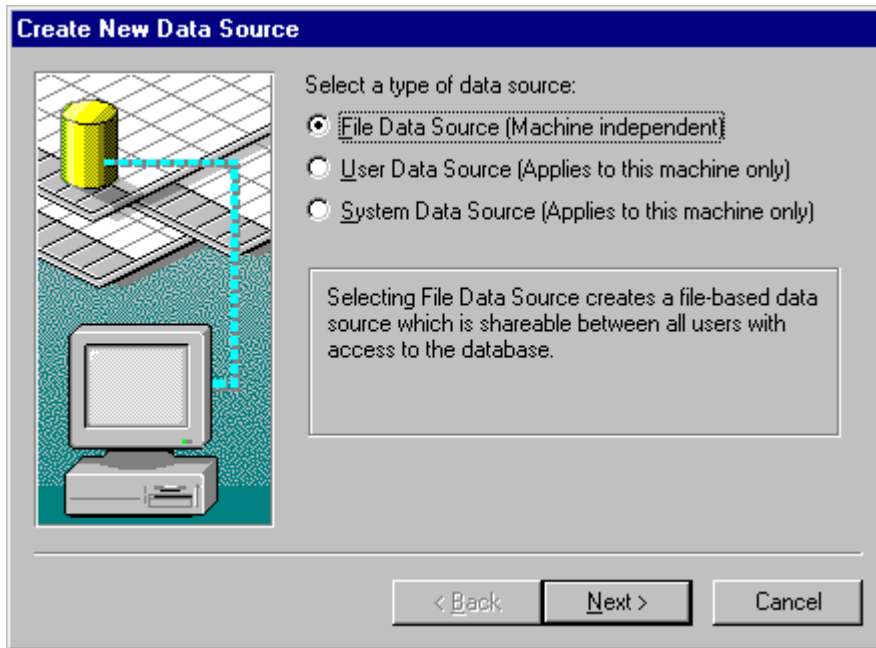
When **SQLCreateDataSource** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------------|---|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwnd</i> argument was invalid or NULL. |
| ODBC_ERROR_INVALID_DSN | Invalid DSN | The <i>lpszDS</i> argument contained a string that was invalid for a DSN. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The call to ConfigDSN with the ODBC_ADD_DSN option failed. |
| ODBC_ERROR_LOAD_LIBRARY_FAILED | Could not load the driver or translator setup library | The driver setup library could not be loaded. |
| ODBC_ERROR_USER_CANCELLED | User canceled operation | User canceled creation of a new data source. |
| ODBC_ERROR_CREATE_DSN_FAILED | Could not create the requested DSN | Could not connect to the database; the call to SQLDriverConnect for a File DSN did not return a successful connection |

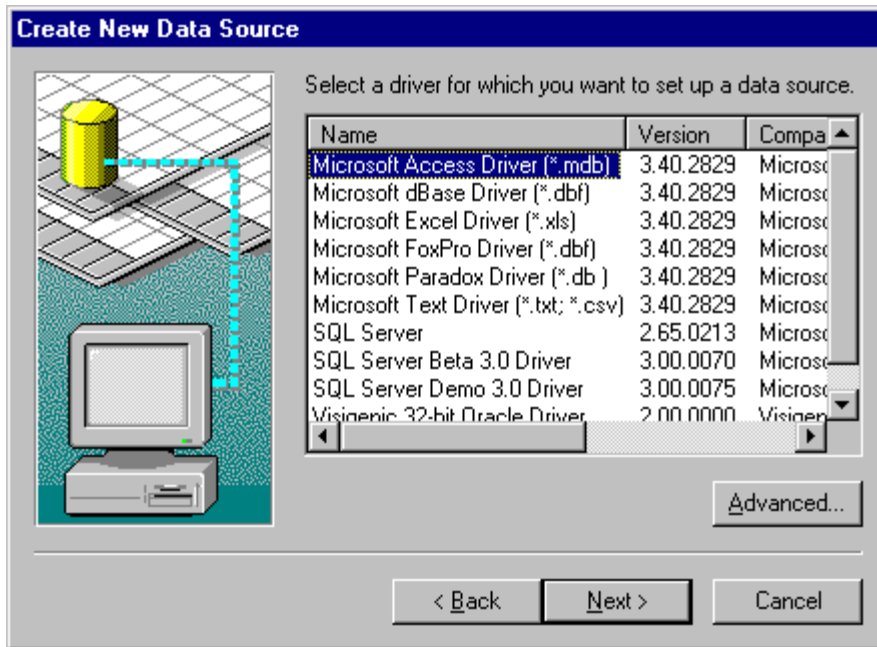
| | | |
|-----------------------|---------------|---|
| | | Could not write to the file. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

If *hwnd* is null, **SQLCreateDataSource** returns FALSE. Otherwise, it displays the Create New Data Source dialog box with a wizard page for choosing the type of data source to be set up:



The default option is File Data Source. When a data source has been chosen, a wizard page containing a list of installed drivers is displayed:

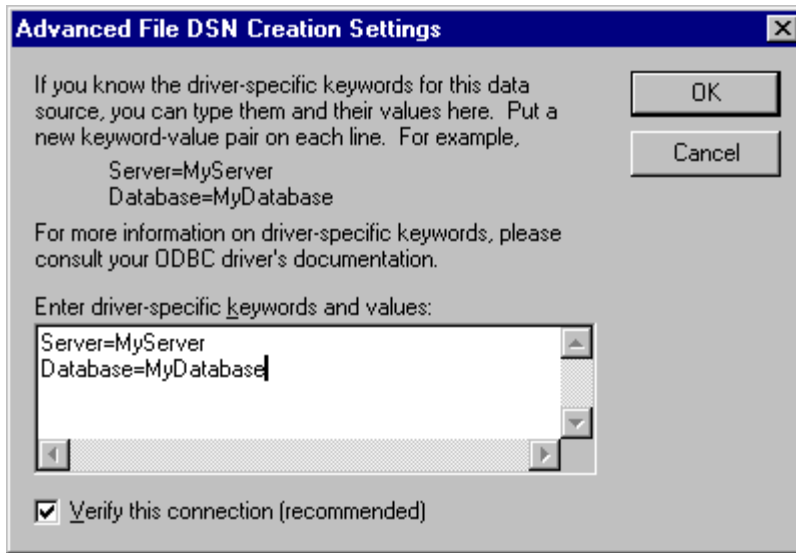


If **Cancel** is clicked, then the dialog box disappears and **SQLCreateDataSource** returns FALSE with the error code of ODBC_ERROR_USER_CANCELED. If either the User Data Source or System Data Source option was selected, then the Advanced button is unavailable.

When the **Next** button is clicked, then one of the following will occur, depending on which type of data source was selected:

- If File Data Source was selected, then a wizard page is displayed for the user to enter a file name.
- If either User Data Source or System Data Source was selected, then a wizard page displaying the type of data source and driver is displayed for review, and when Finish is clicked, the data source is set up.

If Advanced is clicked from the driver list wizard page, a wizard page is displayed for the user to enter driver-specific information. In the text box of this dialog box, type the driver and keywords separated by returns, as follows:

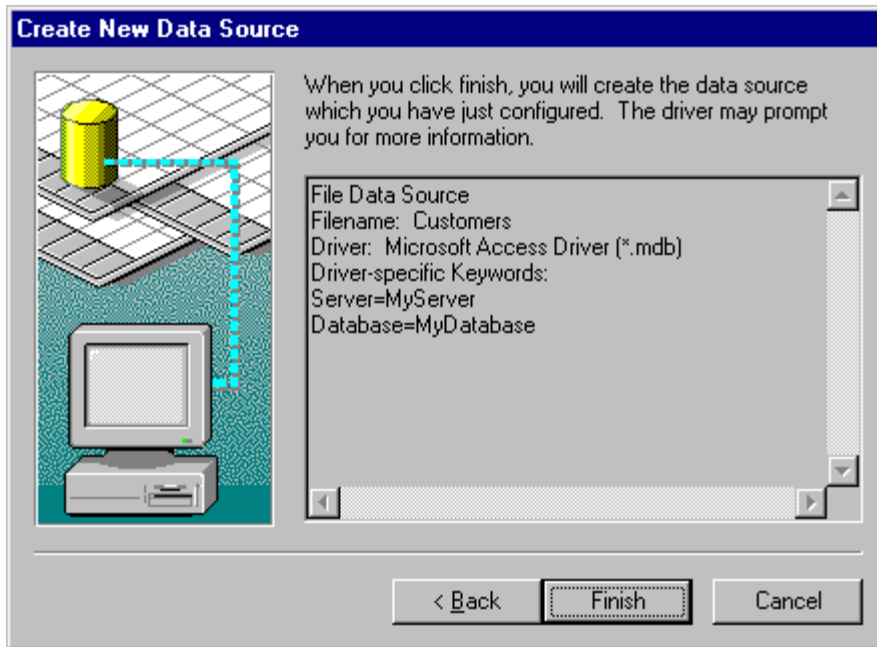


Additional driver-specific keywords can be found under the description of **SQLDriverConnect**. All but DSN are allowed.

The default for the Verify This Connection option is TRUE. This default applies regardless of whether this wizard page is activated. If OK is clicked, then the string specified in the text box and the Verify This Connection option value are cached. (If the X button or Cancel is clicked, then newly entered driver-specific information is lost, because the string specified in the text box and the Verify This Connection option value are not cached.)

If File Data Source was selected in the first wizard page, then after a driver has been selected and the keyword values have been entered in the Advanced wizard page, the user is prompted to enter a file name. Click Browse to search for a file name, in which case the default directory in the Browse box is specified by a combination of the path specified by CommonFileDir in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion and "ODBC\DataSources". (If CommonFileDir was "C:\Program Files\Common Files", the default directory would be "C:\Program Files\Common Files\ODBC\Data Sources".)

When a file name has been entered and **Next** is clicked, then the file name entered is checked for validity against the standard file-naming rules of the operating system. If the file name is invalid, then an error message box notifies the user that an invalid file name was entered. After the user acknowledges the message box, the focus is returned to the wizard page in which the file name is entered. If the file name is valid, then a wizard page showing the selected keyword-value pairs is displayed for review.



If **Finish** is clicked, and File Data Source was selected as the data source type, and the Verify this connection option is TRUE, then **SQLDriverConnect** is called with the SAVEFILE and DRIVER keywords. The DriverCompletion argument is set to SQL_DRIVER_COMPLETE. The file name for the SAVEFILE keyword is the name that was entered or chosen, and, and the driver name for the DRIVER keyword is the name that was chosen. If a driver-specific connection string was specified in the Advanced wizard page, then that string is appended after the DRIVER keyword.

If **SQLDriverConnect** returns SQL_SUCCESS, then the Driver Manager has created the File DSN. SQLCreateDataSource returns TRUE. If **SQLDriverConnect** does not return SQL_SUCCESS, then a warning message box indicates that a connection could not be made to the data source. A DSN with minimal connection information can still be created. This message box allows the user to either cancel or continue with the File DSN creation.

If the user chooses to continue creating the DSN, then this process continues as if the Verify this connection option were set to FALSE. If the user chooses to cancel, then FALSE is returned for **SQLCreateDataSource** with an error code of ODBC_ERROR_CREATE_DSN_FAILED.

If File Data Source was selected as the data source type, and the Verify this connection option is FALSE, then a File DSN is created with the DRIVER keyword and user-specified connect string (if any) from the Advanced wizard page. If the file creation was successful, TRUE is returned for **SQLCreateDataSource**. If the file creation was not successful, then an error message box notifies the user with whatever error was returned from the operating system. FALSE is returned for **SQLCreateDataSource** with an error code of ODBC_ERROR_CREATE_DSN_FAILED. For more information on file data sources, see “Connecting Using File Data Sources” of Chapter 6, “Connecting to a Data Source or Driver,” in the Part I PDF file or see SQLDriverConnect in the Part II PDF file, “ODBC API Reference.” Both files are available on the SOLID Web site.

If User or System Data Source was selected as the data source type, then **ConfigDSN** in the driver setup library is called with the ODBC_ADD_DSN *fRequest*. For more information, see **ConfigDSN** in [“Chapter 22: Setup DLL API Reference.”](#)

Related Functions

| For information about | See |
|-----------------------|--------------------------------------|
| Managing data sources | SQLManageDataSources |

SQLGetConfigMode

Conformance

Version Introduced: ODBC 3.0

Summary

SQLGetConfigMode retrieves the configuration mode that indicates where the ODBC.INI entry listing DSN values is in the system information.

Syntax

```
BOOL SQLGetConfigMode(  
    UWORD * pwConfigMode);
```

Arguments

pwConfigMode

[Output]

Pointer to the buffer containing the configuration mode (see “Comments”). The value in **pwConfigMode* can be:

ODBC_USER_DSN

ODBC_SYSTEM_DSN

ODBC_BOTH_DSN

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLGetConfigMode** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|-----------------------|---------------|---|
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

This function is used to determine where the ODBC.INI entry listing DSN values is in the system information. If **pwConfigMode* is ODBC_USER_DSN, the DSN is a User DSN and the function reads from the ODBC.INI entry in HKEY_CURRENT_USER. If it is ODBC_SYSTEM_DSN, the DSN is a System DSN and the function reads from the ODBC.INI entry in HKEY_LOCAL_MACHINE. If it is ODBC_BOTH_DSN, HKEY_CURRENT_USER is tried, and if it fails, then HKEY_LOCAL_MACHINE is used.

By default, **SQLGetConfigMode** returns ODBC_BOTH_DSN. When a User DSN or a System DSN is created by a call to **SQLConfigDataSource**, the function sets the configuration mode to ODBC_USER_DSN or ODBC_SYSTEM_DSN to distinguish user and System DSNs while modifying a DSN. Prior to returning, **SQLConfigDataSource** resets the configuration mode to ODBC_BOTH_DSN.

Related Functions

| For information about | See |
|--------------------------------|----------------------------------|
| Setting the configuration mode | SQLSetConfigMode |

SQLGetInstalledDrivers

Conformance

Version Introduced: ODBC 1.0

Summary

SQLGetInstalledDrivers reads the [ODBC Drivers] section of the system information and returns a list of descriptions of the installed drivers.

Syntax

```
BOOL SQLGetInstalledDrivers(  
    LPSTR  lpzBuf,  
    WORD   cbBufMax,  
    WORD *  pcbBufOut);
```

Arguments

lpzBuf

[Output]

List of descriptions of the installed drivers. For information about the list structure, see “Comments.”

cbBufMax

[Input]

Length of *lpzBuf*.

pcbBufOut

[Output]

Total number of bytes (excluding the null-termination byte) returned in *lpzBuf*. If the number of bytes available to return is greater than or equal to *cbBufMax*, the list of driver descriptions in *lpzBuf* is truncated to *cbBufMax* minus the null-termination character. The *pcbBufOut* argument can be a null pointer.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLGetInstalledDrivers** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|-----------------------------|-------------------------|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFF_LEN | Invalid buffer length | The <i>lpzBuf</i> argument was NULL or invalid, or the <i>cbBufMax</i> argument was less than or equal to 0. |

| | | |
|--------------------------------|---------------------------------|---|
| ODBC_ERROR_COMPONENT_NOT_FOUND | Component not found in registry | The installer could not find the [ODBC Drivers] section in the registry. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

Each driver description is terminated with a null byte and the entire list is terminated with a null byte (that is, two null bytes mark the end of the list). If the allocated buffer is not large enough to hold the entire list, the list is truncated without error. An error is returned if a null pointer is passed in as *lpzBuf*.

Related Functions

| For information about | See |
|--|---|
| Returning driver descriptions and attributes | SQLDrivers in the Part I PDF file, “ODBC Function Reference,” available on the SOLID Web site. |

SQLGetTranslator

Conformance

Version Introduced: ODBC 2.0

Summary

SQLGetTranslator displays a dialog box from which a user can select a translator.

Syntax

```
BOOL SQLGetTranslator(  
    HWND hwndParent,  
    LPSTR lpzName,  
    WORD cbNameMax,  
    WORD * pcbNameOut,  
    LPSTR lpzPath,  
    WORD cbPathMax,  
    WORD * pcbPathOut,  
    DWORD * pvOption);
```

Arguments

hwndParent

[Input]

Parent window handle.

lpzName

[Input/Output]

Name of the translator from the system information.

cbNameMax

[Input]

Maximum length of the *lpzName* buffer.

pcbNameOut

[Input/Output]

Total number of bytes (excluding the null-termination byte) passed or returned in *lpzName*. If the number of bytes available to return is greater than or equal to *cbNameMax*, the translator name in *lpzName* is truncated to *cbNameMax* minus the null-termination character. The *pcbNameOut* argument can be a null pointer.

lpzPath

[Output]

Full path of the translation DLL.

cbPathMax

[Input]

Maximum length of the *lpszPath* buffer.

pcbPathOut

[Output]

Total number of bytes (excluding the null-termination byte) returned in *lpszPath*. If the number of bytes available to return is greater than or equal to *cbPathMax*, the translation DLL path in *lpszPath* is truncated to *cbPathMax* minus the null-termination character. The *pcbPathOut* argument can be a null pointer.

pvOption

[Output]

32-bit translation option.

Returns

The function returns TRUE if it is successful, FALSE if it fails or the user cancels the dialog box.

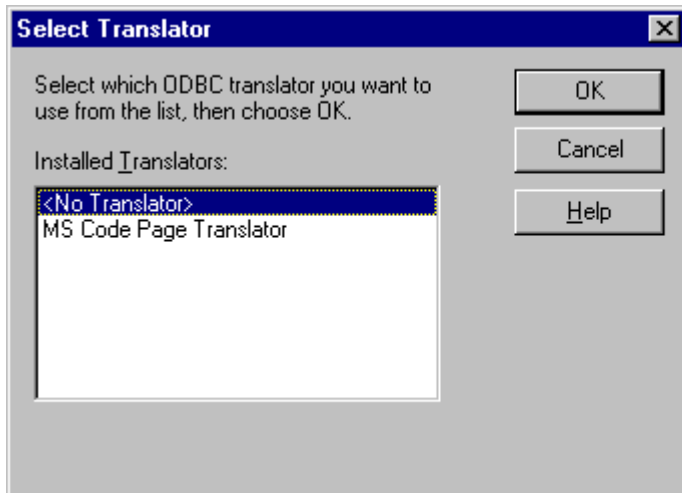
Diagnostics

When **SQLGetTranslator** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------------|---|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFF_LEN | Invalid buffer length | The <i>cbNameMax</i> or <i>cbPathMax</i> argument was less than or equal to 0. |
| ODBC_ERROR_INVALID_HWND | Invalid window handle | The <i>hwndParent</i> argument was invalid or NULL. |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszName</i> argument was invalid. It could not be found in the registry. |
| ODBC_ERROR_LOAD_LIBRARY_FAILED | Could not load the driver or translator setup library | The translator library could not be loaded. |
| ODBC_ERROR_INVALID_OPTION | Invalid transaction option | The <i>pvOption</i> argument contained an invalid value. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

If *hwndParent* is null, or *lpszName*, *lpszPath*, or *pvOption* is a null pointer, **SQLGetTranslator** returns FALSE. Otherwise, it displays the list of installed translators in the following dialog box:



If *lpszName* contains a valid translator name, it is selected. Otherwise, <No Translator> is selected.

If the user chooses <No Translator>, the contents of *lpszName*, *lpszPath*, and *pvOption* are not touched. **SQLGetTranslator** sets *pcbNameOut* and *pcbPathOut* to 0 and returns TRUE.

If the user chooses a translator, **SQLGetTranslator** calls **ConfigTranslator** in the translator's setup DLL. If **ConfigTranslator** returns FALSE, **SQLGetTranslator** returns to its dialog box. If **ConfigTranslator** returns TRUE, **SQLGetTranslator** returns TRUE, along with the selected translator name, path, and translation option.

Related Functions

| For information about | See |
|---------------------------------|---|
| Configuring a translator | ConfigTranslator |
| Getting a translation attribute | SQLGetConnectAttr in the Part II PDF file, "ODBC Function Reference," contained on the SOLID Web site. |
| Setting a translation attribute | SQLSetConnectAttr in the Part II PDF file, "ODBC Function Reference," contained on the SOLID Web site. |

SQLGetPrivateProfileString

Conformance

Version Introduced: ODBC 2.0

Summary

SQLGetPrivateProfileString gets a list of names of values or data corresponding to a value of the system information.

Syntax

```
int SQLGetPrivateProfileString(  
    LPCSTR lpszSection,  
    LPCSTR lpszEntry,  
    LPCSTR lpszDefault,  
    LPCSTR RetBuffer,  
    INT cbRetBuffer,  
    LPCSTR lpszFilename);
```

Arguments

lpszSection

[Input]

Points to a null-terminated string that specifies the section containing the key name. If this argument is NULL, the function copies all section names in the file to the supplied buffer.

lpszEntry

[Input]

Points to the null-terminated string containing the key name whose associated string is to be retrieved. If this argument is NULL, all key names in the section specified by the *lpszSection* argument are copied to the buffer specified by the *RetBuffer* argument.

lpszDefault

[Input]

Points to a null-terminated string that specifies the default value for the given key if the key cannot be found in the initialization file. This argument cannot be NULL.

RetBuffer

[Output]

Points to the buffer that receives the retrieved string.

cbRetBuffer

[Input]

Specifies the size, in characters, of the buffer pointed to by the *RetBuffer* argument.

lpszFilename

[Output]

Points to a null-terminated string that names the initialization file. If this argument does not contain a full path to the file, the default directory is searched.

Returns

SQLGetPrivateProfileString returns an integer value that indicates the number of characters read.

Diagnostics

When a call to **SQLGetPrivateProfileString** fails, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLGetPrivateProfileString is provided as a simple way to port drivers and driver setup DLLs from Windows to Windows NT. Calls to **GetPrivateProfileString** that retrieve a profile string from the ODBC.INI file should be replaced with calls to **SQLGetPrivateProfileString**.

SQLGetPrivateProfileString calls functions in the Win32 API to retrieve the requested names of values or data corresponding to a value of the ODBC.INI subkey of the system information.

The configuration mode (as set by **SQLSetConfigMode**) indicates where the ODBC.INI entry listing DSN values is in the system information. If the DSN is a User DSN (the configuration mode is **USERDSN_ONLY**), the function reads from the ODBC.INI entry in **HKEY_CURRENT_USER**. If the DSN is a System DSN (**SYSTEMDSN_ONLY**), the function reads from the ODBC.INI entry in **HKEY_LOCAL_MACHINE**. If the configuration mode is **BOTHDSN**, **HKEY_CURRENT_USER** is tried, and if it fails, then **HKEY_LOCAL_MACHINE** is used.

Related Functions

| For information about | See |
|---|--|
| Writing a value to the system information | SQLWritePrivateProfileString |

SQLInstallDriverEx

Conformance

Version Introduced: ODBC 3.0

Summary

SQLInstallDriverEx adds information about the driver to the ODBCINST.INI entry in the system information and increments the driver's *UsageCount* by 1. However, if a version of the driver already exists, but the *UsageCount* value for the driver does not exist, the new *UsageCount* value is set to 2.

This function does not actually copy any files. It is the responsibility of the calling program to copy the driver's files to the target directory properly.

Syntax

```
BOOL SQLInstallDriverEx(  
    LPCSTR  lpszDriver,  
    LPCSTR  lpszPathIn,  
    LPSTR   lpszPathOut,  
    WORD    cbPathOutMax,  
    WORD *  pcbPathOut,  
    WORD    fRequest,  
    LPDWORD lpdwUsageCount);
```

Arguments

lpszDriver

[Input]

The driver description (usually the name of the associated DBMS) presented to users instead of the physical driver name. The *lpszDriver* argument must contain a list of keyword-value pairs describing the driver.

For more information, see "Comments."

lpszPathIn

[Input]

Full path of the target directory of the installation, or a null pointer. If *lpszPathIn* is a null pointer, the drivers will be installed in the system directory.

lpszPathOut

[Output]

Path of the target directory where the driver should be installed. If the driver has not previously been installed, then *lpszPathOut* should be the same as *lpszPathIn*. If the driver was previously installed, then *lpszPathOut* is the path of the previous installation.

cbPathOutMax

[Input]

Length of *lpzPathOut*.

pcbPathOut

[Output]

Total number of bytes (excluding the null-termination character) available to return in *lpzPathOut*. If the number of bytes available to return is greater than or equal to *cbPathOutMax*, then the output path in *lpzPathOut* is truncated to *cbPathOutMax* minus the null-termination character. The *pcbPathOut* argument can be a null pointer.

fRequest

[Input]

Type of request. The *fRequest* argument must contain one of the following values:

ODBC_INSTALL_INQUIRY: Inquire about where a driver can be installed.

ODBC_INSTALL_COMPLETE: Complete the installation request.

lpdwUsageCount

[Output]

The usage count of the driver after this function has been called.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLInstallDriverEx** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|----------------------------------|-----------------------------|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFF_LEN | Invalid buffer length | The <i>lpzPathOut</i> argument was not large enough to contain the output path. The buffer contains the truncated path. The <i>cbPathOutMax</i> argument was 0 and <i>fRequest</i> was ODBC_INSTALL_COMPLETE. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request | The <i>fRequest</i> argument was not one of the following: ODBC_INSTALL_INQUIRY ODBC_INSTALL_COMPLETE |
| ODBC_ERROR_INVALID_KEYWORD_VALUE | Invalid keyword-value pairs | The <i>lpzDriver</i> argument contained a syntax error. |
| ODBC_ERROR_ | Invalid install path | The <i>lpzPathIn</i> argument contained an invalid path. |

| | | |
|-----------------------------------|--|---|
| INVALID_PATH | | |
| ODBC_ERROR_LOAD_LIBRARY_FAILED | Could not load the driver or translator setup library | The driver setup library could not be loaded. |
| ODBC_ERROR_INVALID_PARAM_SEQUENCE | Invalid parameter sequence | The <i>lpszDriver</i> argument did not contain a list of keyword-value pairs. |
| ODBC_ERROR_USAGE_UPDATE_FAILED | Could not increment or decrement the component usage count | The installer failed to increment the driver's usage count. |

Comments

The *lpszDriver* argument is a list of attributes in the form of keyword-value pairs. Each pair is terminated with a null byte and the entire list is terminated with a null byte (that is, two null bytes mark the end of the list). The format of this list is:

```
driver-desc\0Driver=driver-DLL-filename\0[Setup=setup-DLL-filename\0]
[driver-attr-keyword1=value1\0][driver-attr-keyword2=value2\0]...\0
```

where \0 is a null byte and *driver-attr-keywordn* is any driver attribute keyword described in “Driver Keyword Sections” in [“Chapter 18: Installing ODBC Components”](#). The keywords must appear in the specified order. For example, suppose a driver for formatted text files has separate driver and setup DLLs and can use files with the .txt and .csv extensions. The *lpszDriver* argument for this driver might be:

```
Text\0Driver=TEXT.DLL\0Setup=TXTSETUP.DLL\0FileUsage=1\0
FileExtns=*.txt,*.csv\0\0
```

Suppose that a driver for SQL Server does not have a separate setup DLL and does not have any driver attribute keywords. The *lpszDriver* argument for this driver might be:

```
SQL Server\0Driver=SQLSRVR.DLL\0\0
```

After **SQLInstallDriverEx** retrieves information about the driver from the *lpszDriver* argument, it adds the driver description to the [ODBC Drivers] section of the ODBCINST.INI entry in the system information. It then creates a section titled with the driver's description and adds the full paths of the driver DLL and the setup DLL. Finally, it returns the path of the target directory of the installation but does not copy the driver files to it. The calling program must actually copy the driver files to the target directory.

SQLInstallDriverEx increments the component usage count for the installed driver by 1. If a version of the driver already exists, but the component usage count for the driver does not exist, the new component usage count value is set to 2.

The application setup program is responsible for physically copying the driver file, and maintaining the file usage count. If the driver file has not previously been installed, the application setup program must copy the file in the *lpszPathIn* path, and create the file usage count. If the file has previously been installed, the setup program merely increments the file usage count, and returns the path of the prior installation in the *lpszPathOut* argument.

Note For more information about component usage counts and file usage counts, see “Usage Counting” in Chapter 18, “Installing ODBC Components.”

If an older version of the driver file was previously installed by the application, the driver should be uninstalled, then reinstalled, so that the driver component usage count is valid. **SQLConfigDriver** (with an

fRequest of ODBC_REMOVE_DRIVER) should first be called, then **SQLRemoveDriver** should be called to decrement the component usage count. **SQLInstallDriverEx** should then be called to reinstall the driver, incrementing the component usage count. The application setup program must physically replace the old file with the new file. The file usage count will remain the same, and any other application that used the older version file will now use the newer version.

Note If the driver was previously installed, and **SQLInstallDriverEx** is called to install the driver in a different directory, the function will return TRUE, but *lpszPathOut* will include the directory where the driver was already installed. It will not include the directory entered in the *lpszDriver* argument.

The length of the path in *lpszPathOut* in **SQLInstallDriverEx** allows for a two-phase install process, so an application can determine what *cbPathOutMax* should be by calling **SQLInstallDriverEx** with an *fRequest* of ODBC_INSTALL_INQUIRY mode. This will return the total number of bytes available in the *pcbPathOut* buffer. **SQLInstallDriverEx** can then be called with an *fRequest* of ODBC_INSTALL_COMPLETE and the *cbPathOutMax* argument set to the value in the *pcbPathOut* buffer, plus the null-termination character.

If you choose not to use the two-phase model for **SQLInstallDriverEx**, then you must set *cbPathOutMax*, which defines the size of the storage for the path of the target directory, to the value _MAX_PATH, as defined in STDLIB.H, to prevent truncation.

When *fRequest* is ODBC_INSTALL_COMPLETE, **SQLInstallDriverEx** does not allow *lpszPathOut* to be NULL (or *cbPathOutMax* to be 0). If *fRequest* is ODBC_INSTALL_COMPLETE, FALSE is returned when the number of bytes available to return is greater than or equal to *cbPathOutMax*, with the result that truncation occurs.

After **SQLInstallDriverEx** has been called, and the application setup program has copied the driver file (if necessary), the driver setup DLL must call **SQLConfigDriver** to set the configuration for the driver.

Related Functions

| For information about | See |
|-------------------------------|--------------------------------------|
| Installing the Driver Manager | SQLInstallDriverManager below |

SQLInstallDriverManager

Conformance

Version Introduced: ODBC 1.0

Summary

SQLInstallDriverManager returns the path of the target directory for the installation of the ODBC core components. The calling program must actually copy the Driver Manager's files to the target directory.

Syntax

```
BOOL SQLInstallDriverManager(  
    LPSTR  lpszPath,  
    WORD   cbPathMax,  
    WORD *  pcbPathOut);
```

Arguments

lpszPath

[Output]

Path of the target directory of the installation.

cbPathMax

[Input]

Length of *lpszPath*. This must be at least _MAX_PATH bytes.

pcbPathOut

[Output]

Total number of bytes (excluding the null-termination byte) returned in *lpszPath*. If the number of bytes available to return is greater than or equal to *cbPathMax*, the path in *lpszPath* is truncated to *cbPathMax* minus the null-termination character. The *pcbPathOut* argument can be a null pointer.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLInstallDriverManager** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|-----------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFF_LEN | Invalid buffer length | The <i>lpszPath</i> argument was not large enough to contain the output path. The buffer contains the truncated path. |

| | | |
|--------------------------------|--|---|
| | | The <i>cbPathMax</i> argument was less than <code>_MAX_PATH</code> . |
| ODBC_ERROR_USAGE_UPDATE_FAILED | Could not increment or decrement the component usage count | The installer failed to increment the ODBC core component usage count. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLInstallDriverManager is called to return the path for ODBC core components and increment the component usage count in the system information. If a version of the Driver Manager already exists, but the component usage count for the driver does not exist, the new component usage count value is set to 2.

The application setup program is responsible for physically copying the core component files, and maintaining the file usage counts. If a core component file has not previously been installed, the application setup program must copy the file, and create the file usage count. If the file has previously been installed, the setup program merely increments the file usage count.

If an older version of the Driver Manager was previously installed by the application setup program, the core components should be uninstalled, then reinstalled, so that the core component usage count is valid.

SQLRemoveDriverManager should first be called to decrement the component usage count.

SQLInstallDriverManager should then be called to increment the component usage count. The application setup program must physically replace the old core component files with the new files. The file usage counts will remain the same, and other applications that used the older version core component files will now use the newer version files.

In a fresh install of the ODBC core components, drivers, and translators, the application setup program should call the following functions in sequence: **SQLInstallDriverManager**, **SQLInstallDriverEx**, **SQLConfigDriver** (with an *fRequest* of `ODBC_INSTALL_DRIVER`), then **SQLInstallTranslatorEx**. In an uninstall of the core components, drivers, and translators, the application setup program should call the following functions in sequence: **SQLRemoveTranslator**, **SQLRemoveDriver**, then **SQLRemoveDriverManager**. These functions must be called in this sequence. In an upgrade of all components, all the uninstall functions should be called in sequence, then all the install functions should be called in sequence.

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a driver | SQLConfigDriver |
| Installing a driver | SQLInstallDriverEx |
| Installing a translator | SQLInstallTranslatorEx |
| Removing a driver | SQLRemoveDriver |
| Removing the Driver Manager | SQLRemoveDriverManager |
| Removing a translator | SQLRemoveTranslator |

SQLInstallerError

Conformance

Version Introduced: ODBC 3.0

Summary

SQLInstallerError returns error or status information for the ODBC installer functions.

Syntax

```
RETCODE SQLInstallerError(  
    WORD    iError,  
    DWORD * pfErrorCode,  
    LPSTR   lpzErrorMsg,  
    WORD    cbErrorMsgMax,  
    WORD *  pcbErrorMsg);
```

Arguments

iError

[Input]

Error record number. Valid numbers are from 1 to 8.

pfErrorCode

[Output]

Installer error code. (For more information, see “Comments.”)

lpzErrorMsg

[Output]

Pointer to storage for the error message text.

cbErrorMsgMax

[Input]

Maximum length of the *szErrorMsg* buffer. This must be less than or equal to SQL_MAX_MESSAGE_LENGTH minus the null-termination character.

cbErrorMsgMax

[Input]

Maximum length of the *szErrorMsg* buffer. This must be less than or equal to SQL_MAX_MESSAGE_LENGTH minus the null-termination character.

pcbErrorMsg

[Output]

Pointer to the total number of bytes (excluding the null-termination character) available to return in *lpzErrorMsg*. If the number of bytes available to return is greater than or equal to *cbErrorMsgMax*, the

error message text in *lpszErrorMsg* is truncated to *cbErrorMsgMax* minus the null-termination character bytes. The *pcbErrorMsg* argument can be a null pointer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, or SQL_ERROR.

Diagnostics

SQLInstallerError does not post error values for itself. **SQLInstallerError** returns SQL_NO_DATA when it is unable to retrieve any error information (in which case *pfErrorCode* is undefined). If **SQLInstallerError** cannot access error values for any reason that would normally return SQL_ERROR, **SQLInstallerError** returns SQL_ERROR, but does not post any error values. If either the *lpszErrorMsg* argument is NULL, the *cbErrorMsgMax* is less than or equal to 0, then this function returns SQL_ERROR. If the buffer for the error message is too short, **SQLInstallerError** returns SQL_SUCCESS_WITH_INFO, and returns the correct *pfErrorCode* value for **SQLInstallerError**.

To determine whether a truncation occurred in the error message, an application can compare the value in the *cbErrorMsgMax* argument to the actual length of the message text written to the *pcbErrorMsg* argument. If truncation does occur, the correct buffer length should be allocated for *lpszErrorMsg* and **SQLInstallerError** should be called again with the corresponding *iError* record.

Comments

An application calls **SQLInstallerError** when a previous call to the ODBC installer function returns FALSE. ODBC installer and driver or translator setup functions post zero or more errors only when the function fails (returns FALSE); therefore, an application calls **SQLInstallerError** only after an ODBC installer function fails.

The ODBC installer error queue is flushed each time a new installer function is called. Therefore, an application cannot expect to retrieve errors for functions other than from the last installer function call.

To retrieve multiple errors for a function call, an application calls **SQLInstallerError** multiple times.

When there is no additional information, **SQLInstallerError** returns SQL_NO_DATA, the *pfErrorCode* argument is undefined, the *pcbErrorMsg* argument equals 0, and the *lpszErrorMsg* argument contains a single null-termination character (unless the *cbErrorMsgMax* argument is equal to 0).

SQLInstallTranslator

Conformance

Version Introduced: ODBC 2.5, Deprecated

Summary

In ODBC 3.0, **SQLInstallTranslator** has been replaced by [SQLInstallTranslatorEx](#). Calls to **SQLInstallTranslator** will be mapped to **SQLInstallTranslatorEx**. For more information, see **SQLInstallTranslatorEx**.

SQLInstallTranslator will return FALSE if an application calls it in the ODBC 3.x Driver Manager with the *lpszInfFile* argument set to a value other than NULL. The ODBC.INF file used in ODBC 2.x is no longer supported in ODBC 3.x, even for backward compatibility.

SQLInstallTranslatorEx

Conformance

Version Introduced: ODBC 3.0

Summary

SQLInstallTranslatorEx adds information about a translator to the ODBCINST.INI section of the system information.

Syntax

```
BOOL SQLInstallTranslatorEx (  
    LPCSTR lpzTranslator,  
    LPCSTR lpzPathIn,  
    LPSTR lpzPathOut,  
    WORD cbPathOutMax,  
    WORD * pcbPathOut,  
    WORD fRequest,  
    LPDWORD lpdwUsageCount);
```

Arguments

lpzTranslator

[Input]

This must contain a list of keyword-value pairs describing the translator. The Translator and Setup keywords have to be included in the *lpzTranslator* string. The translation DLL is listed with the Translator keyword, and the translator setup DLL is listed with the Setup keyword. Each pair is terminated with a NULL byte, and the entire list is terminated with a NULL byte (that is, two NULL bytes mark the end of the list). The format of *lpzTranslator* is:

```
translator-desc  
\0Translator=translator-DLL-filename\0[Setup=setup-DLL-filename\0]\0
```

lpzPathIn

[Input]

Full path of where the translator is to be installed or a null pointer. If *lpzPath* is a null pointer, then the translators will be installed in the System directory.

lpzPathOut

[Output]

The path of the target directory where the translator should be installed. If the translator has never been installed, then *lpzPathOut* is the same as *lpzPathIn*. If there exists a prior installation of the translator, then *lpzPathOut* is the path of the prior installation.

cbPathOutMax

[Input]

Length of *lpzPathOut*.

pcbPathOut

[Output]

Total number of bytes available to return in *lpszPathOut*. If the number of bytes available to return is greater than or equal to *cbPathOutMax*, the output path in *lpszPathOut* is truncated to *pcbPathOutMax* minus the null-termination character. The *pcbPathOut* argument can be a null pointer.

fRequest

[Input]

Type of request. *fRequest* must contain one of the following values:

ODBC_INSTALL_INQUIRY: Inquire about where a translator can be installed.

ODBC_INSTALL_COMPLETE: Complete the installation request.

lpdwUsageCount

[Output]

The usage count of the translator after this function has been called.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLInstallTranslatorEx** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|----------------------------------|-----------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFFER_LEN | Invalid buffer length | The <i>lpszPath</i> argument was not large enough to contain the output path. The buffer contains the truncated path. The <i>cbPathOutMax</i> argument was 0 and the <i>fRequest</i> argument was ODBC_INSTALL_COMPLETE. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request | The <i>fRequest</i> argument was not one of the following: ODBC_INSTALL_INQUIRY ODBC_INSTALL_COMPLETE |
| ODBC_ERROR_INVALID_KEYWORD_VALUE | Invalid keyword-value pairs | The <i>lpszTranslator</i> argument contained a syntax error |
| ODBC_ERROR_INVALID_PATH | Invalid install path | The <i>lpszPathIn</i> argument contained an invalid path. |
| ODBC_ERROR_ | Invalid parameter | The <i>lpszTranslator</i> argument did not contain a list |

| | | |
|--|---|---|
| INVALID_ PARAM_SEQUENCE | sequence | of keyword-value pairs. |
| ODBC_ERROR_ USAGE_ UPDATE_ FAILED | Could not increment or decrement the registry's component usage count | The installer failed to increment the translator's usage count. |

Comments

SQLInstallTranslatorEx provides a mechanism to install just the translator. This function does not actually copy any files. The calling program is responsible for copying the translator files.

SQLInstallTranslatorEx increments the component usage count for the installed translator by 1. If a version of the translator already exists, but the component usage count for the translator does not exist, the new component usage count value is set to 2.

The application setup program is responsible for physically copying the translator file, and maintaining the file usage count. If the translator file has not previously been installed, the application setup program must copy the file or files, and create the file or files usage count. If the file has previously been installed, the setup program simply increments the file usage count.

If an older version of the translator was previously installed by the application, the translator should be uninstalled, then reinstalled, so that the translator component usage count is valid. **SQLRemoveTranslator** should be called to decrement the component usage count, and then **SQLInstallTranslatorEx** should then be called to increment the component usage count. The application setup program must physically replace the old file or files with the new file. The file usage count will remain the same, and other applications that used the older version file will now use the newer version.

The length of the path in *lpszPathOut* in **SQLInstallTranslatorEx** allows for a two-phase install process, so an application can determine what *cbPathOutMax* should be by calling **SQLInstallTranslatorEx** with an *fRequest* of ODBC_INSTALL_INQUIRY mode. This will return the total number of bytes available in the *pcbPathOut* buffer. **SQLInstallTranslatorEx** can then be called with an *fRequest* of ODBC_INSTALL_COMPLETE and the *cbPathOutMax* argument set to the value in the *pcbPathOut* buffer, plus the null-termination character.

If you choose not to use the two-phase model for **SQLInstallTranslatorEx**, then you must set *cbPathOutMax*, which defines the size of the storage for the path of the target directory, to the value `_MAX_PATH`, as defined in `STDLIB.H`, to prevent truncation.

When *fRequest* is ODBC_INSTALL_COMPLETE, **SQLInstallTranslatorEx** does not allow *lpszPathOut* to be NULL (or *cbPathOutMax* to be 0). If *fRequest* is ODBC_INSTALL_COMPLETE, FALSE is returned when the number of bytes available to return is greater than or equal to *cbPathOutMax*, with the result that truncation occurs.

Related Functions

| For information about | See |
|--|-------------------------------------|
| Returning a default translation option | ConfigTranslator |
| Selecting translators | SQLGetTranslator |
| Removing translators | SQLRemoveTranslator |

SQLManageDataSources

Conformance

Version Introduced: ODBC 2.0

Summary

SQLManageDataSources displays a dialog box with which users can set up, add, and delete data sources in the system information.

Syntax

```
BOOL SQLManageDataSources(  
    HWND hwnd);
```

Arguments

hwnd

[Input]
Parent window handle.

Returns

SQLManageDataSources returns FALSE if *hwnd* is not a valid window handle. Otherwise, it returns TRUE.

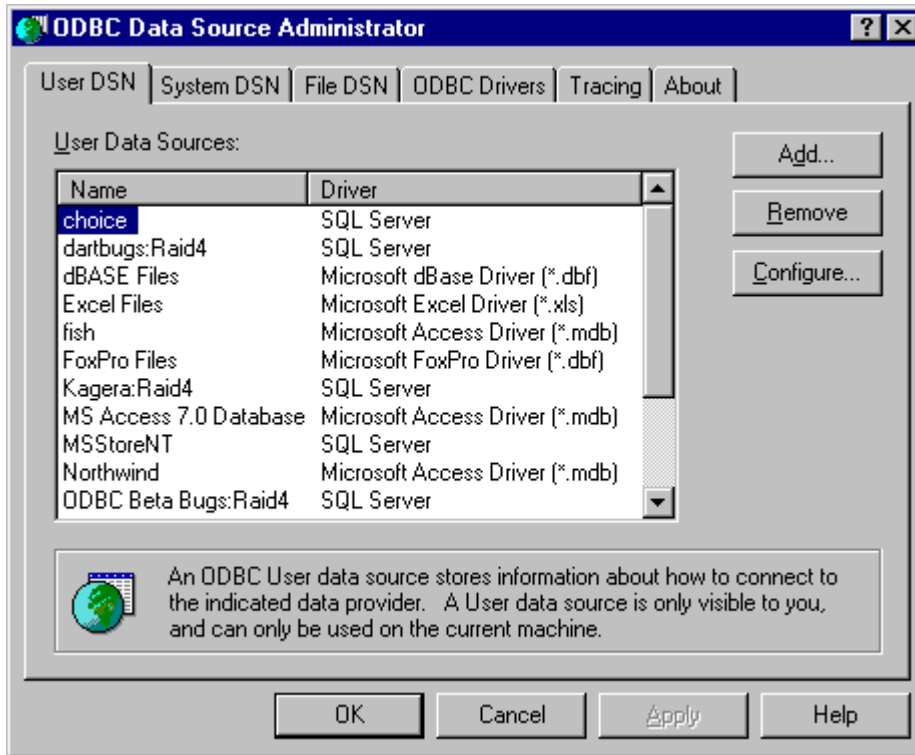
Diagnostics

When **SQLManageDataSources** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|---------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_REQUEST_FAILED | <i>Request</i> failed | The call to ConfigDSN failed. |
| ODBC_ERROR_INVALID__HWND | Invalid window handle | The <i>hwnd</i> argument was invalid or NULL. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

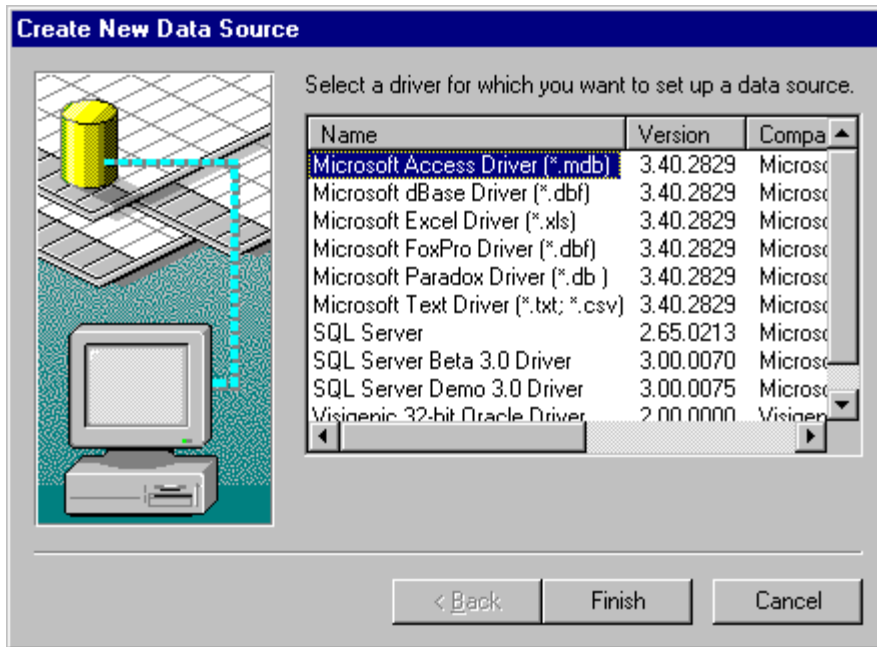
Managing Data Sources

SQLManageDataSources initially displays the ODBC Data Source Administrator dialog box.



The dialog box displays the data sources listed in the system information under three tabs: **User DSN**, **System DSN**, and **File DSN**. If the user double-clicks a data source or selects a data source and clicks **Configure**, **SQLManageDataSources** calls **ConfigDSN** in the setup DLL with the ODBC_CONFIG_DSN option.

If the user clicks **Add**, **SQLManageDataSources** displays the **Create New Data Source** dialog box, shown in the following figure.



The dialog box displays a list of installed drivers. If the user double-clicks a driver or selects a driver and clicks **OK**, **SQLManageDataSources** calls **ConfigDSN** in the setup DLL and passes it the ODBC_ADD_DSN option.

If the user selects a data source and clicks **Remove**, **SQLManageDataSources** asks if the user wants to delete the data source. If the user clicks **Yes**, **SQLManageDataSources** calls **ConfigDSN** in the setup DLL with the ODBC_REMOVE_DSN option.

The **Add Data Source** dialog box is used to add or delete either a user data source, a system data source, or a file data source.

System DSNs

The **Create New Data Source** dialog box allows you to add a system data source to your local computer or delete one, or to set the configuration for a system data source.

A data source set up with a system data-source name (DSN) can be used by more than one user on the same machine. It can also be used by a system-wide service, which can then gain access to the data source even if no user is logged onto the machine.

A System DSN is registered in the HKEY_LOCAL_MACHINE entry in the system information, rather than the HKEY_CURRENT_USER entry. It is not tied to one user who logs on with his or her particular user name and password, but can be used by any user of that machine, or by an automatic system-wide service. The System DSN is, however, tied to one machine. It does not support the capability of using remote DSNs between machines. System DSNs are registered as follows in the system information:

```
HKEY_LOCAL_MACHINE
SOFTWARE
  ODBC
    ODBC.INI
```

User DSNs

DSNs created for individual users will be called User DSNs, to distinguish them from System DSNs. User DSNs are registered as follows in the system information:

```
HKEY_CURRENT_USER
SOFTWARE
  ODBC
    ODBC.INI
```

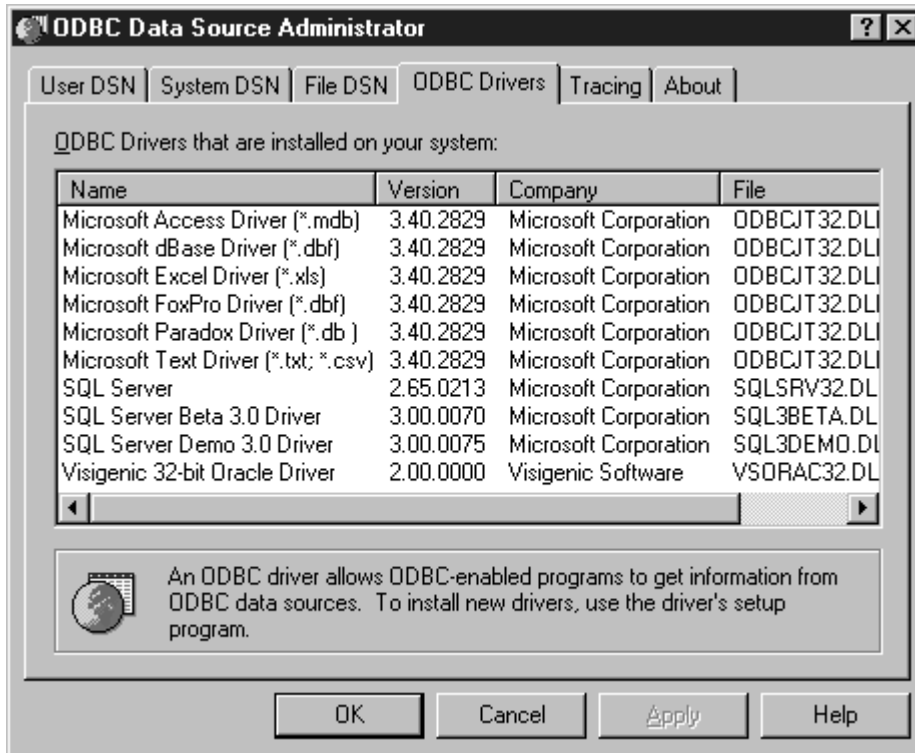
File DSNs

A file data source does not have a data source name, as does a machine data source, and is not registered to any one user or machine. The connection information for that data source is contained in a .dsn file that can be copied to any machine. A file data source can be shareable, in which case the .dsn file resides on a network and can be used simultaneously by multiple users on the network as long as the user has the appropriate driver installed. A file data source can also be unshareable, in which case it can only be used on a single machine.

For more information on file data sources, see “Connecting Using File Data Sources” of Chapter 6, “Connecting to a Data Source or Driver,” in the Part I PDF file, or see **SQLDriverConnect** in the Part II PDF file. Both are available on the SOLID Web site.

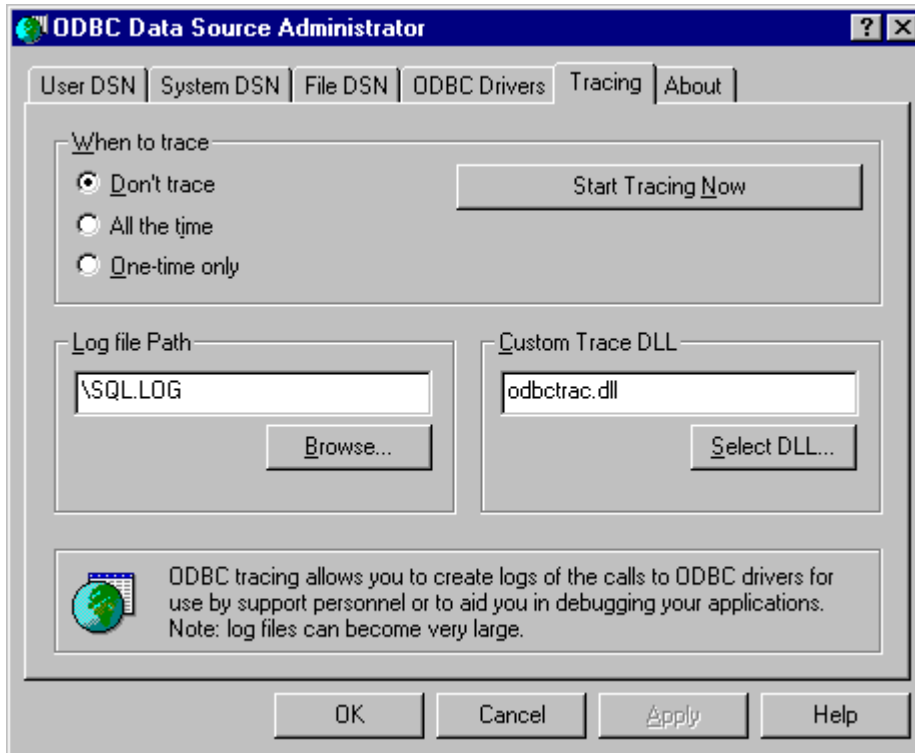
Managing Drivers

If the user clicks the **ODBC Drivers** tab in the **ODBC Data Source Administrator** dialog box, **SQLManageDataSources** displays a list of ODBC driver installed on the system, and information about the drivers. The date displayed is the creation date of the driver.



Tracing Options

If the user clicks the **Tracing** tab in the **ODBC Data Source Administrator** dialog box, **SQLManageDataSources** displays tracing options.



If the user selects or clears any of the check boxes in the **When to trace** section of the tab page, and clicks **OK**, **SQLManageDataSources** sets the **Trace** keyword in the [ODBC] section of the system information to 1 or 0 accordingly. If **All the time** is chosen, tracing is performed for each subsequent connection; if **One-time only** is chosen, tracing is performed for the duration of the connection only.

If the user clicks **Start Tracing Now**, and then clicks **OK**, **SQLManageDataSources** enables tracing manually for all applications currently running on the machine.

If the user specifies the name of a trace file in the **Log file Path** text box, then clicks **OK**, **SQLManageDataSources** sets the **TraceFile** keyword in the [ODBC] section of the system information to the specified name.

For more information on tracing, see “Tracing” in Chapter 17, “Programming Considerations” in the Part I PDF file available on the SOLID Web site. For more information about the **Trace**, **TraceAutoStop**, and **TraceFile** keywords, see “[ODBC Core Subkey](#)” in Chapter 19, “Configuring Data Sources.”

Related Functions

| For information about | See |
|-----------------------|-------------------------------------|
| Creating data sources | SQLCreateDataSource |

SQLPostInstallerError

Conformance

Version Introduced: ODBC 3.0

Summary

SQLPostInstallerError provides a mechanism for a driver or translator setup library to report errors for the **ConfigDriver**, **ConfigDSN**, and **ConfigTranslator** functions to the installer error queue. Applications do not use this API; they use **SQLInstallerError** to retrieve the error.

Syntax

```
RETCODE SQLPostInstallerError (  
    DWORD fErrorCode,  
    LPSTR szErrorMsg);
```

Arguments

fErrorCode

[Input]
Installer error code.

szErrorMsg

[Input]
Error message text.

Returns

SQL_SUCCESS or SQL_ERROR.

Diagnostics

SQLPostInstallerError does not post error values for itself. If the error was successfully posted to the installer error queue (retrievable using **SQLInstallerError**), SQL_SUCCESS is returned. SQL_ERROR will be returned if the value in the *dwErrorCode* argument is not one of the specified installer error codes.

Related Functions

| For information about | See |
|---|----------------------------------|
| Adding, modifying, or removing a driver | ConfigDriver |
| Adding, modifying, or removing data sources | ConfigDSN |
| Setting a translation option | ConfigTranslator |

SQLReadFileDSN

Conformance

Version Introduced: ODBC 3.0

Summary

SQLReadFileDSN reads information from a File DSN.

Syntax

```
BOOL SQLReadFileDSN(  
    LPCSTR  lpszFileName,  
    LPCSTR  lpszAppName,  
    LPCSTR  lpszKeyName,  
    LPSTR   lpszString,  
    WORD    cbString,  
    WORD *   pcbString);
```

Arguments

lpszFileName

[Input]

Pointer to the data buffer containing the name of the .dsn file. A .dsn extension is appended to all file names that do not already have a .dsn extension. The value in **lpszFileName* must be a null-terminated string.

lpszAppName

[Input]

Pointer to the data buffer containing the name of the application. This is “ODBC” for the ODBC section. The value in **lpszAppName* must be a null-terminated string.

lpszKeyName

[Input]

Pointer to the data buffer containing the name of the key to be read. See “Comments” for reserved keywords. The value in **lpszAppName* must be a null-terminated string.

lpszString

[Output]

Pointer to the data buffer containing the string associated with the key to be read.

If **lpszFileName* is a valid .dsn file name, but the *lpszAppName* argument is a null pointer and the *lpszKeyName* argument is a null pointer, then **lpszString* contains a list of valid applications. If **lpszFileName* is a valid .dsn file name, and **lpszAppName* is a valid application name, but the *lpszKeyName* argument is a null pointer, then **lpszString* contains a list of valid reserved keywords in the appropriate section of the DSN file, delimited by semicolons. If **lpszFileName* is a valid .dsn file name, but **lpszAppName* is a null pointer, and the *lpszKeyName* argument is a null pointer, then **lpszString* contains a list of the sections in the DSN file, delimited by semicolons.

cbString

[Input]

Length of the **lpszString* buffer.

pcbString

[Output]

Total number of bytes available to return in **lpszString*. If the number of bytes available to return is greater than or equal to *cbString*, the output string in **lpszString* is truncated to *cbString* minus the null-termination character. The *pcbString* argument can be a null pointer.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLReadFileDSN** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|------------------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_BUFF_LEN | Invalid buffer length | The <i>lpszString</i> argument was NULL. The <i>cbString</i> argument was less than or equal to 0. |
| ODBC_ERROR_INVALID_PATH | Invalid install path | The path of the file name specified in the <i>lpszFileName</i> argument was invalid. |
| ODBC_ERROR_INVALID_REQUEST_TYPE | Invalid type of request | The <i>lpszAppName</i> argument was NULL while the <i>lpszKeyName</i> argument was valid. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |
| ODBC_ERROR_OUTPUT_STRING_TRUNCATED | Output string truncated | The string returned in <i>*lpszString</i> was truncated because the value in <i>cbString</i> was less than or equal to the value in <i>*pcbString</i> . |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The keyword did not exist in the file DSN. |

Comments

ODBC reserves the section name [ODBC] in which to store the connection information. The reserved keywords for this section are the same as those reserved for a connect string in **SQLDriverConnect** (for more information, see the **SQLDriverConnect** function description).

Applications may use these reserved keywords to read the information in a File DSN. If an applications wants to find out the DSN-less connection string associated with a File DSN, it can call **SQLReadFileDSN** for any of the reserved connection string keywords in the [ODBC] section. The full connection string

passed in a DSN-less connection is a combination of all of the keywords (reserved and driver-specific) in the [ODBC] section.

Related Functions

| For information about | See |
|-----------------------------------|---------------------------------|
| Writing information to a File DSN | SQLWriteFileDSN |

SQLRemoveDefaultDataSource

Conformance

Version Introduced: ODBC 1.0, Deprecated

Summary

In ODBC 3.0, the **SQLRemoveDefaultDataSource** function has been replaced by a call to **SQLConfigDataSource** with an *fRequest* argument of ODBC_REMOVE_DEFAULT_DSN. If an ODBC 2.x installation program calls this function, the ODBC installer will map it to the following **SQLConfigDataSource** call:

SQLConfigDataSource (NULL, ODBC_REMOVE_DEFAULT_DSN, NULL, NULL)

SQLRemoveDriver

Conformance

Version Introduced: ODBC 3.0

Summary

SQLRemoveDriver changes or removes information about the driver from the ODBCINST.INI entry in the system information.

Syntax

```
BOOL SQLRemoveDriver (  
    LPCSTR lpszDriver,  
    BOOL fRemoveDSN,  
    LPDWORD lpdwUsageCount);
```

Arguments

lpszDriver

[Input]

The name of the driver as registered in the ODBCINST.INI key of the system information.

fRemoveDSN

[Input]

The valid values are:

TRUE:

Remove DSNs associated with the driver specified in *lpszDriver*.

FALSE:

Do not remove DSNs associated with the driver specified in *lpszDriver*.

lpdwUsageCount

[Output]

The usage count of the driver after this function has been called.

Returns

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE.

Diagnostics

When **SQLRemoveDriver** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|---------------------|-------------------------|---|
| ODBC_ERROR_ | General installer error | An error occurred for which there was no specific |

| | | |
|--------------------------------|--|---|
| GENERAL_ERR | | installer error. |
| ODBC_ERROR_COMPONENT_NOT_FOUND | Component not found in registry | The installer could not remove the driver information because it either did not exist in the registry or could not be found in the registry. |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszDriver</i> argument was invalid. |
| ODBC_ERROR_USAGE_UPDATE_FAILED | Could not increment or decrement the component usage count | The installer failed to decrement the usage count of the driver. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The <i>fRemoveDSN</i> argument was TRUE; however, one or more DSNs could not be removed. The call to SQLConfigDriver with the ODBC_REMOVE_DRIVER request failed. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLRemoveDriver complements the [SQLInstallDriverEx](#) function, and updates the component usage count in the system information. This function should only be called from a setup application.

SQLRemoveDriver will decrement the component usage count value by 1. If the component usage count goes to 0, then the following will occur:

1. The **SQLConfigDriver** function with the ODBC_REMOVE_DRIVER option will be called. If the *fRemoveDSN* option is set to TRUE, the **ConfigDSN** function calls **SQLRemoveDSNFromIni** to remove all the data sources associated with the driver specified in *lpszDriver*. If the *fRemoveDSN* option is set to FALSE, the data sources will not be deleted.
2. The driver entry in the system information will be removed. The driver entry is in the following system information location, under the driver name:

```
HKEY_LOCAL_MACHINE
    SOFTWARE
        ODBC
            ODBCINST.INI
```

SQLRemoveDriver does not actually remove any files. The calling program is responsible for deleting files, and maintaining the file usage count. Only after both the component usage count and the file usage count have reached zero is a file physically deleted. Some files in a component can be deleted, and others not deleted, depending upon whether the files are used by other applications that have incremented the file usage count.

SQLRemoveDriver is also called as part of an upgrade process. If an application detects that it has to perform an upgrade, and it has previously installed the driver, then the driver should be removed, then reinstalled. **SQLRemoveDriver** should first be called to decrement the component usage count, then **SQLInstallDriverEx** should then be called to increment the component usage count. The application setup program must physically replace the old files with the new files. The file usage count will remain the same, and other applications that use the older version files will now use the newer version.

Related Functions

| For information about | See |
|---|---|
| Adding, modifying, or removing a driver | ConfigDriver (in the Setup DLL) |
| Adding, modifying, or removing a driver | SQLConfigDriver |
| Installing a driver | SQLInstallDriverEx |

SQLRemoveDriverManager

Conformance

Version Introduced: ODBC 3.0

Summary

SQLRemoveDriverManager changes or removes information about the ODBC core components from the ODBCINST.INI entry in the system information.

Syntax

```
BOOL SQLRemoveDriverManager (  
    LPDWORD pdwUsageCount);
```

Arguments

pdwUsageCount

[Output]

The usage count of the Driver Manager after this function has been called.

Returns

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE.

Diagnostics

When **SQLRemoveDriverManager** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------------|--|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_COMPONENT_NOT_FOUND | Component not found in registry | The installer could not remove the Driver Manager information because it either did not exist in the registry or could not be found in the registry. |
| ODBC_ERROR_USAGE_UPDATE_FAILED | Could not increment or decrement the component usage count | The installer failed to decrement the usage count of the Driver Manager. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLRemoveDriverManager complements the **SQLInstallDriverManager** function, and updates the component usage count in the system information. This function should only be called from a setup application.

SQLRemoveDriverManager will decrement the core component usage count by 1. If the component usage count goes to 0, then the entry system information will be removed. The core component entry is in the following location in the system information, under the title "ODBC Core":

```
HKEY_LOCAL_MACHINE
    SOFTWARE
        ODBC
            ODBCINST.INI
```

Caution An application should not physically remove Driver Manager files when the component usage count and the file usage count reach zero.

SQLRemoveDriverManager does not actually remove any files. The calling program is responsible for deleting files and maintaining the file usage counts. Driver Manager files should not, however, be removed when both the component usage count and the file usage count have reached zero, because these files may be used by other applications that have not incremented the file usage count.

SQLRemoveDriverManager is called as part of the Uninstall process. ODBC core components (which include the Driver Manager, Cursor Library, Installer, Language Library, Administrator, thunking files, and so on) are uninstalled as a whole. The following files are not removed when **SQLRemoveDriverManager** is called as part of the Uninstall process:

| | |
|--------------|--------------|
| ODBC32DLL | ODBCCP32.DLL |
| ODBCCR32.DLL | ODBC16GT.DLL |
| ODBCCU32.DLL | ODBC32GT.DLL |
| ODBCINT.DLL | DS16GT.DLL |
| ODBCTRAC.DLL | DS32GT.DLL |
| MSVCRT40.DLL | ODBCAD32.EXE |
| ODBCCP32.CPL | |

SQLRemoveDriverManager is also called as part of an upgrade process. If an application detects that it has to perform an upgrade, and it has previously installed the driver, then the driver should be removed, and then reinstalled. **SQLRemoveDriverManager** should first be called to decrement the component usage count. **SQLInstallDriverEx** should then be called to increment the component usage count. The application setup program must physically replace the old core component files with the new files. The file usage counts will remain the same, and other applications that use the older version core component files will now use the newer version files.

Related Functions

| | |
|------------------------------|---|
| For information about | See |
| Installing a Driver Manager | SQLInstallDriverManager |

SQLRemoveDSNFromIni

Conformance

Version Introduced: ODBC 1.0

Summary

SQLRemoveDSNFromIni removes a data source from the system information.

Syntax

```
BOOL SQLRemoveDSNFromIni(  
    LPCSTR lpszDSN);
```

Arguments

lpszDSN

[Input]

Name of the data source to remove.

Returns

The function returns TRUE if it removes the data source or the data source was not in the ODBC.INI file. It returns FALSE if it fails to remove the data source.

Diagnostics

When **SQLRemoveDSNFromIni** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|---------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_DSN | Invalid DSN | The <i>lpszDSN</i> argument was invalid. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The installer could not remove the DSN info from the registry. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLRemoveDSNFromIni removes the data source name from the [ODBC Data Sources] section of the system information. It also removes the data source specification section from the system information.

This function should only be called from a driver setup library.

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a driver | ConfigDSN |
| Adding, modifying, or removing a data source | SQLCreateDataSource |
| Removing the default data source | SQLRemoveDefaultDataSource |
| Adding a data source name to the system information | SQLWriteDSNToIni |

SQLRemoveTranslator

Conformance

Version Introduced: ODBC 3.0

Summary

SQLRemoveTranslator removes information about a translator from the ODBCINST.INI section of the system information and decrements the translator's component usage count by 1.

Syntax

```
BOOL SQLRemoveTranslator (  
    LPCSTR lpszTranslator,  
    LPDWORD lpdwUsageCount);
```

Arguments

lpszTranslator

[Input]

The name of the translator as registered in the ODBCINST.INI key of the system information.

lpdwUsageCount

[Output]

The usage count of the translator after this function has been called.

Returns

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE.

Diagnostics

When **SQLRemoveTranslator** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------------|---|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_COMPONENT_NOT_FOUND | Component not found in registry | The installer could not remove the translator information because it either did not exist in the registry or could not be found in the registry |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszTranslator</i> argument was invalid. |
| ODBC_ERROR_USAGE_UPDATE_FAILED | Could not increment or decrement the component usage count. | The installer failed to decrement the usage count of the driver. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLRemoveTranslator complements the **SQLInstallTranslatorEx** function, and updates the component usage count in the system information. This function should only be called from a setup application.

SQLRemoveTranslator will decrement the component usage count by 1. If the component usage count goes to 0, then the translator entry in the system information will be removed. The translator entry is in the following location in the system information, under the translator name:

```
HKEY_LOCAL_MACHINE
    SOFTWARE
        ODBC
            ODBCINST.INI
```

SQLRemoveTranslator does not actually remove any files. The calling program is responsible for deleting files, and maintaining the file usage count. Only after both the component usage count and the file usage count have reached zero is a file physically deleted. Some files in a component can be deleted, and others not deleted, depending upon whether the files are used by other applications that have incremented the file usage count.

SQLRemoveTranslator is also called as part of an upgrade process. If an application detects that it has to perform an upgrade, and it has previously installed the driver, then the driver should be removed, then reinstalled. **SQLRemoveTranslator** should first be called to decrement the component usage count, then **SQLInstallTranslatorEx** should then be called to increment the component usage count. The application setup program must physically replace the old files with the new files. The file usage count will remain the same, and other applications that use the older version files will now use the newer version.

Related Functions

| For information about | See |
|-------------------------|--|
| Installing a translator | SQLInstallTranslatorEx |

SQLSetConfigMode

Conformance

Version Introduced: ODBC 3.0

Summary

SQLSetConfigMode sets the configuration mode that indicates where the ODBC.INI entry listing DSN values is in the system information.

Syntax

```
BOOL SQLSetConfigMode(  
    WORD wConfigMode);
```

Arguments

wConfigMode

[Input]

The installer configuration mode (see “Comments”). The value in *wConfigMode* can be:

ODBC_USER_DSN
ODBC_SYSTEM_DSN
ODBC_BOTH_DSN

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLSetConfigMode** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|-----------------------------------|----------------------------|---|
| ODBC_ERROR_INVALID_PARAM_SEQUENCE | Invalid parameter sequence | The <i>wConfigMode</i> argument did not contain ODBC_USER_DSN, ODBC_SYSTEM_DSN, or ODBC_BOTH_DSN. |

Comments

This function is used to set where the ODBC.INI entry listing DSN values is in the system information. If *wConfigMode* is ODBC_USER_DSN, the DSN is a User DSN and the function reads from the ODBC.INI entry in HKEY_CURRENT_USER. If it is ODBC_SYSTEM_DSN, the DSN is a System DSN and the function reads from the ODBC.INI entry in HKEY_LOCAL_MACHINE. If it is ODBC_BOTH_DSN, HKEY_CURRENT_USER is tried, and if it fails, then HKEY_LOCAL_MACHINE is used.

This function does not affect **SQLCreateDataSource** and **SQLDriverConnect**. The configuration mode has to be set when a driver reads from the registry by calling **SQLGetPrivateProfileString** or writes to the registry by calling **SQLWritePrivateProfileString**. Calls to **SQLGetPrivateProfileString** and **SQLWritePrivateProfileString** use the configuration mode to know which part of the registry to operate on.

Caution **SQLSetConfigMode** should only be called when necessary; if the mode is improperly set, the ODBC Installer may fail to function properly.

SQLSetConfigMode makes a direct registry modification of the configuration mode. This is apart from the process of modifying the configuration mode by a call to **SQLConfigDataSource**. A call to **SQLConfigDataSource** sets the configuration mode to distinguish user and System DSNs when modifying a DSN. Prior to returning, **SQLConfigDataSource** resets the configuration mode to BOTHDSN.

Related Functions

For information about

Creating a data source

Connecting to a data source using a connection string or dialog box

Retrieving the configuration mode

See

[SQLCreateDataSource](#)

SQLDriverConnect in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site.

[SQLGetConfigMode](#)

SQLValidDSN

Conformance

Version Introduced: ODBC 2.0

Summary

SQLValidDSN checks the length and validity of the data source name before the name is added to the system information.

Syntax

```
BOOL SQLValidDSN(  
    LPCSTR lpszDSN);
```

Arguments

lpszDSN

[Input]

Data source name to be checked.

Returns

The function returns TRUE if the data source name is valid. It returns FALSE if the data source name is invalid or the function call failed.

Diagnostics

When **SQLValidDSN** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. A **pfErrorCode* is returned only if the function call failed, not if FALSE was returned because the data source name is invalid. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLValidDSN is called by a driver's **ConfigDSN** to check the length of the data source name, and the validity of the individual characters in the data source name. It checks whether the length of the name is greater than SQL_MAX_DSN_LENGTH, as defined in SQLEXT.H. (Note that the length of the data source name is also checked by **SQLWriteDSNTToIni**.) **SQLValidDSN** checks whether any of the following invalid characters are included in the data source name:

[] { } () , ; ? * = ! @ \

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a data source | ConfigDSN (in the Setup DLL) |
| Adding, modifying, or removing a data sources | SQLConfigDataSource |
| Writing a data source name to the system | SQLWriteDSNTToIni |

| | |
|-------------|--|
| information | |
|-------------|--|

SQLWriteDSNToIni

Conformance

Version Introduced: ODBC 1.0

Summary

SQLWriteDSNToIni adds a data source to the system information.

Syntax

```
BOOL SQLWriteDSNToIni(  
    LPCSTR lpszDSN,  
    LPCSTR lpszDriver);
```

Arguments

lpszDSN

[Input]

Name of the data source to add.

lpszDriver

[Input]

Driver description (usually the name of the associated DBMS) presented to users instead of the physical driver name.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLWriteDSNToIni** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|---------------------------|-----------------------------------|--|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_DSN | Invalid DSN | The <i>lpszDSN</i> argument contained a string that was invalid for a DSN. |
| ODBC_ERROR_INVALID_NAME | Invalid driver or translator name | The <i>lpszDriver</i> argument was invalid. |
| ODBC_ERROR_REQUEST_FAILED | Request failed | The installer failed to create a DSN in the registry. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLWriteDSNToIni adds the data source to the [ODBC Data Sources] section of the system information. It then creates a specification section for the data source and adds a single keyword (Driver) with the name

of the driver DLL as its value. If the data source specification section already exists, **SQLWriteDSNToIni** removes the old section before creating the new one.

The caller of this function must add any driver-specific keywords and values to the data source specification section of the system information.

If the name of the data source is Default, **SQLWriteDSNToIni** also creates the default driver specification section in the system information.

This function should only be called from a setup DLL.

Related Functions

| For information about | See |
|---|--|
| Adding, modifying, or removing a data source | ConfigDSN (in the Setup DLL) |
| Adding, modifying, or removing a data sources | SQLConfigDataSource |
| Removing a data source name from the system information | SQLRemoveDSNFromIni |

SQLWriteFileDSN

Conformance

Version Introduced: ODBC 3.0

Summary

SQLWriteFileDSN writes information to a File DSN.

Syntax

```
BOOL SQLWriteFileDSN(  
    LPCSTR  lpzFileName,  
    LPCSTR  lpzAppName,  
    LPCSTR  lpzKeyName,  
    LPCSTR  lpzString);
```

Arguments

lpzFileName

[Input]

Pointer to the name of the File DSN. A DSN extension is appended to all file names that do not already have a DSN extension.

lpzAppName

[Input]

Pointer to the name of the application. This is “ODBC” for the ODBC section.

lpzKeyName

[Input]

Pointer to the name of the key to be read. See “Comments” for reserved keywords.

lpzString

[Output]

Pointed to the string associated with the key to be written. The maximum length of the string pointed to by this argument is 32,767 bytes.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLWriteFileDSN** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|-------------------------|-------------------------|---|
| ODBC_ERROR_GENERAL_ERR | General installer error | An error occurred for which there was no specific installer error. |
| ODBC_ERROR_INVALID_PATH | Invalid install path | The path of the file name specified in the <i>lpzFileName</i> argument was invalid. |
| ODBC_ERROR_INVALID | Invalid type of request | The <i>lpzAppName</i> , <i>lpzKeyName</i> , or <i>lpzString</i> |

| | |
|----------------|--------------------|
| D_REQUEST_TYPE | argument was NULL. |
|----------------|--------------------|

Comments

ODBC reserves the section name [ODBC] in which to store the connection information. The reserved keywords for this section are the same as those reserved for a connect string in **SQLDriverConnect** (for more information, see the **SQLDriverConnect** function description).

Applications may use these reserved keywords to write information directly to a File DSN. If an application wants to create or modify the DSN-less connection string associated with a File DSN, it can call **SQLWriteFileDSN** for any of the reserved connection string keywords in the [ODBC] section.

If the *lpszString* argument is a null pointer, the keyword pointed to by the *lpszKeyName* argument will be deleted from the .dsn file. If the *lpszString* and *lpszKeyName* arguments are both null pointers, the section pointed to by the *lpszAppName* argument will be deleted from the .dsn file.

Related Functions

| For information about | See |
|------------------------------------|--------------------------------|
| Reading information from File DSNs | SQLReadFileDSN |

SQLWritePrivateProfileString

Conformance

Version Introduced: ODBC 2.0

Summary

SQLWritePrivateProfileString writes a value name and data to the ODBC.INI subkey of the system information.

Syntax

```
BOOL SQLWritePrivateProfileString(  
    LPCSTR lpzSection,  
    LPCSTR lpzEntry,  
    LPCSTR lpzString,  
    LPCSTR lpzFilename);
```

Arguments

lpzSection

[Input]

Points to a null-terminated string containing the name of the section to which the string will be copied. If the section does not exist, it is created. The name of the section is case-independent; the string can be any combination of uppercase and lowercase letters.

lpzEntry

[Input]

Points to a null-terminated string containing the name of the key to be associated with a string. If the key does not exist in the specified section, it is created. If this argument is NULL, the entire section, including all entries within the section, is deleted.

lpzString

[Input]

Points to a null-terminated string to be written to the file. If this argument is NULL, the key pointed to by the *lpzEntry* argument is deleted.

lpzFilename

[Output]

Points to a null-terminated string that names the initialization file.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

Diagnostics

When **SQLWritePrivateProfileString** returns FALSE, an associated **pfErrorCode* value may be obtained by calling **SQLInstallerError**. The following table lists the **pfErrorCode* values that can be returned by **SQLInstallerError** and explains each one in the context of this function.

| <i>*pfErrorCode</i> | Error | Description |
|--------------------------|-------------------------|--|
| ODBC_ERROR_ GENERAL_ ERR | General installer error | An error occurred for which there was no specific installer error. |

| | | |
|--------------------------------|----------------|---|
| ODBC_ERROR_REQUEST_FAILED_PATH | Request failed | The requested system information could not be written. |
| ODBC_ERROR_OUT_OF_MEM | Out of memory | The installer could not perform the function because of a lack of memory. |

Comments

SQLWritePrivateProfileString is provided as a simple way to port drivers and driver setup DLLs from Windows to Windows NT. Calls to **WritePrivateProfileString** that write a profile string to the ODBC.INI file should be replaced with calls to **SQLWritePrivateProfileString**. **SQLWritePrivateProfileString** calls functions in the Win32 API to add the specified value name and data to the ODBC.INI subkey of the system information.

A configuration mode indicates where the ODBC.INI entry listing DSN values is in the system information. If the DSN is a User DSN (the state variable is USERDSN_ONLY), the function writes to the ODBC.INI entry in HKEY_CURRENT_USER. If the DSN is a System DSN (SYSTEMDSN_ONLY), the function writes to the ODBC.INI entry in HKEY_LOCAL_MACHINE. If the state variable is BOTHDSN, HKEY_CURRENT_USER is tried, and if it fails, then HKEY_LOCAL_MACHINE is used.

Related Functions

| For information about | See |
|---|--|
| Getting a value from the system information | SQLGetPrivateProfileString |

Chapter 24: Translation DLL Function Reference

This chapter describes the syntax of the translation DLL API, which consists of two functions: **SQLDriverToDataSource** and **SQLDataSourceToDriver**. These functions must be included in the DLL that performs translation for the driver.

SQLDriverToDataSource

SQLDriverToDataSource supports translations for ODBC drivers. This function is not called by ODBC-enabled applications; applications request translation through **SQLSetConnectAttr**. The driver associated with the *ConnectionHandle* specified in **SQLSetConnectAttr** calls the specified DLL to perform translations of all data flowing from the driver to the data source. A default translation DLL can be specified in the ODBC initialization file.

Syntax

```
BOOL SQLDriverToDataSource(  
    UDWORD fOption,  
    SWORD fSqlType,  
    PTR rgbValueIn,  
    SDWORD cbValueIn,  
    PTR rgbValueOut,  
    SDWORD cbValueOutMax,  
    SDWORD * pcbValueOut,  
    UCHAR * szErrorMsg,  
    SWORD cbErrorMsgMax,  
    SWORD * pcbErrorMsg);
```

Arguments

fOption

[Input]

Option value.

fSqlType

[Input]

The ODBC SQL data type. This argument tells the driver how to convert *rgbValueIn* into a form acceptable by the data source. For a list of valid SQL data types, see the “SQL Data Types” section in Appendix D, “Data Types” of the **SOLID Programmer Guide**.

rgbValueIn

[Input]

Value to translate.

cbValueIn

[Input]

Length of *rgbValueIn*.

rgbValueOut

[Output]

Result of the translation.

Note The translation DLL does not null-terminate this value.

cbValueOutMax

[Input]

Length of *rgbValueOut*.

pcbValueOut

[Output]

The total number of bytes (excluding the null-termination byte) available to return in *rgbValueOut*.

For character or binary data, if this is greater than or equal to *cbValueOutMax*, the data in *rgbValueOut* is truncated to *cbValueOutMax* bytes.

For all other data types, the value of *cbValueOutMax* is ignored and the translation DLL assumes that the size of *rgbValueOut* is the size of the default C data type of the SQL data type specified with *fSqlType*.

The *pcbValueOut* argument can be a null pointer.

szErrorMsg

[Output]

Pointer to storage for an error message. This is an empty string unless the translation failed.

cbErrorMsgMax

[Input]

Length of *szErrorMsg*.

pcbErrorMsg

[Output]

Pointer to the total number of bytes (excluding the null-termination byte) available to return in *szErrorMsg*. If this is greater than or equal to *cbErrorMsg*, the data in *szErrorMsg* is truncated to *cbErrorMsgMax* minus the null-termination character. The *pcbErrorMsg* argument can be a null pointer.

Returns

TRUE if the translation was successful.

FALSE if the translation failed.

Comments

The driver calls **SQLDriverToDataSource** to translate *all* data (SQL statements, parameters, and so on) passing from the driver to the data source. The translation DLL may not translate some data, depending on the data's type and the purpose of the translation DLL. For example, a DLL that translates character data from one code page to another ignores all numeric and binary data.

The value of *fOption* is set to the value of *vParam* specified by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_OPTION attribute. It is a 32-bit value that has a specific meaning for a given translation DLL. For example, it could specify a certain character set translation.

If the same buffer is specified for *rgbValueIn* and *rgbValueOut*, the translation of data in the buffer will be performed in place.

Note that, although *cbValueIn*, *cbValueOutMax*, and *pcbValueOut* are of the type SDWORD, **SQLDriverToDataSource** does not necessarily support huge pointers.

If **SQLDriverToDataSource** returns FALSE, data truncation may have occurred during translation. If *pcbValueOut*, the number of bytes available to return in the output buffer, is greater than *cbValueOutMax*, the length of the output buffer, then truncation occurred. The driver must determine whether or not the

truncation was acceptable. If truncation did not occur, the **SQLDriverToDataSource** returned FALSE due to another error. In either case, a specific error message is returned in *szErrorMsg*.

For more information about translating data, see “Translation DLLs” of Chapter 17, “Programming Considerations” in the Part I PDF file available on the SOLID Web site.

Related Functions

| For information about | See |
|---|---|
| Translating data returned from the data source | SQLConfigDataSource |
| Returning the setting of a connection attribute | SQLGetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |
| Setting a connection attribute | SQLSetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |

SQLDataSourceToDriver

SQLDataSourceToDriver supports translations for ODBC drivers. This function is not called by ODBC-enabled applications; applications request translation through **SQLSetConnectAttr**. The driver associated with the *ConnectionHandle* specified in **SQLSetConnectAttr** calls the specified DLL to perform translations of all data flowing from the data source to the driver. A default translation DLL can be specified in the ODBC initialization file.

Syntax

```
BOOL SQLDataSourceToDriver(  
    UDWORD fOption,  
    SWORD fSqlType,  
    PTR rgbValueIn,  
    SDWORD cbValueIn,  
    PTR rgbValueOut,  
    SDWORD cbValueOutMax,  
    SDWORD * pcbValueOut,  
    UCHAR * szErrorMsg,  
    SWORD cbErrorMsgMax,  
    SWORD * pcbErrorMsg);
```

Arguments

fOption

[Input]

Option value.

fSqlType

[Input]

The SQL data type. This argument tells the driver how to convert *rgbValueIn* into a form acceptable by the application. For a list of valid SQL data types, see the “SQL Data Types” section in Appendix D, “Data Types” of the **SOLID Programmer Guide**.

rgbValueIn

[Input]

Value to translate.

cbValueIn

[Input]

Length of *rgbValueIn*.

rgbValueOut

[Output]

Result of the translation.

Note The translation DLL does not null-terminate this value.

cbValueOutMax

[Input]

Length of *rgbValueOut*.

pcbValueOut

[Output]

The total number of bytes (excluding the null-termination byte) available to return in *rgbValueOut*.

For character or binary data, if this is greater than or equal to *cbValueOutMax*, the data in *rgbValueOut* is truncated to *cbValueOutMax* bytes.

For all other data types, the value of *cbValueOutMax* is ignored and the translation DLL assumes that the size of *rgbValueOut* is the size of the default C data type of the SQL data type specified with *fSqlType*.

The *pcbValueOut* argument can be a null pointer.

szErrorMsg

[Output]

Pointer to storage for an error message. This is an empty string unless the translation failed.

cbErrorMsgMax

[Input]

Length of *szErrorMsg*.

pcbErrorMsg

[Output]

Pointer to the total number of bytes (excluding the null-termination byte) available to return in *szErrorMsg*. If this is greater than or equal to *cbErrorMsg*, the data in *szErrorMsg* is truncated to *cbErrorMsgMax* minus the null-termination character. The *pcbErrorMsg* argument can be a null pointer.

Returns

TRUE if the translation was successful.

FALSE if the translation failed.

Comments

The driver calls **SQLDataSourceToDriver** to translate *all* data (result set data, table names, row counts, error messages, and so on) passing from the data source to the driver. The translation DLL may not translate some data, depending on the data's type and the purpose of the translation DLL; for example, a DLL that translates character data from one code page to another ignores all numeric and binary data.

The value of *fOption* is set to the value of *vParam* specified by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_OPTION attribute. It is a 32-bit value which has a specific meaning for a given translation DLL. For example, it could specify a certain character set translation.

If the same buffer is specified for *rgbValueIn* and *rgbValueOut*, the translation of data in the buffer will be performed in place.

Note that, although *cbValueIn*, *cbValueOutMax*, and *pcbValueOut* are of the type SDWORD, **SQLDataSourceToDriver** does not necessarily support huge pointers.

If **SQLDataSourceToDriver** returns FALSE, data truncation may have occurred during translation. If *pcbValueOut*, the number of bytes available to return in the output buffer, is greater than *cbValueOutMax*, the length of the output buffer, then truncation occurred. The driver must determine whether the truncation was acceptable. If truncation did not occur, the **SQLDataSourceToDriver** returned FALSE due to another error. In either case, a specific error message is returned in *szErrorMsg*.

For more information about translating data, see “Translation DLLs” of Chapter 17, “Programming Considerations” in the Part II PDF file, available on the SOLID Web site.

Related Functions

| For information about | See |
|---|---|
| Translating data being sent to the data source | SQLDriverToDataSource |
| Returning the setting of a connection attribute | SQLGetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |
| Setting a connection attribute | SQLSetConnectAttr in the Part II PDF file, “ODBC API Reference” available on the SOLID Web site. |