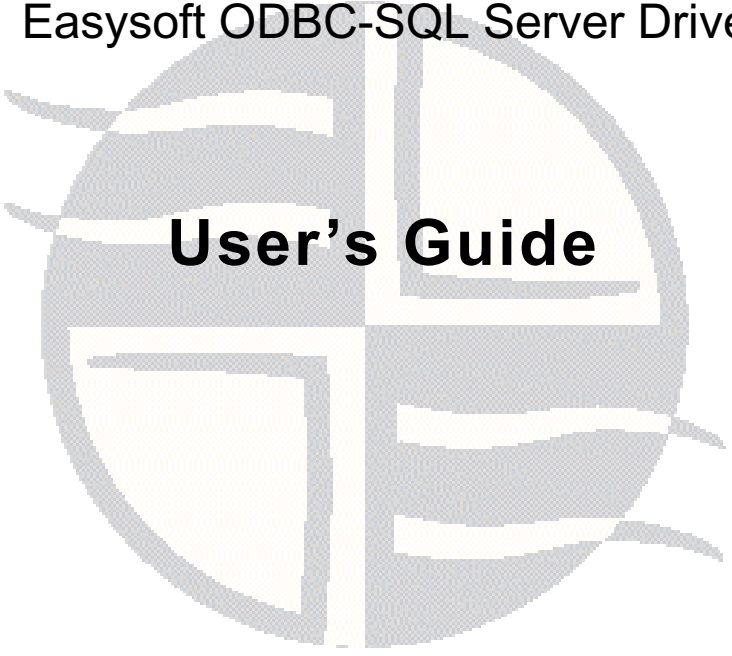
Easysoft Data Access

Easysoft ODBC-SQL Server Driver



User's Guide

This manual documents version 2.1.n of the Easysoft ODBC-SQL Server Driver.

Publisher: Easysoft Limited

Thorp Arch Grange

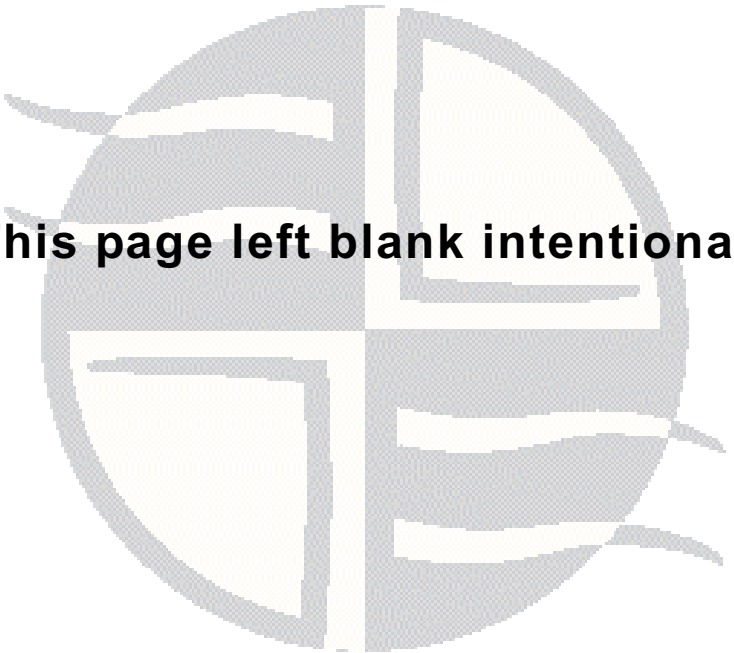
Thorp Arch

Wetherby

LS23 7BA

United Kingdom

LIST OF FIGURES



This page left blank intentionally

PREFACE

About this manual

This manual is intended for use by anyone who wants to install the Easysoft ODBC-SQL Server Driver, configure it, and then access SQL Server data sources from an ODBC-enabled application.

Chapter Guide

Intended Audience

The Unix-based sections require experience of using Unix shell commands. You need to be able to do basic tasks such as editing text files. More complex tasks are described in detail, but it helps to understand how your system handles dynamic linking of shared objects.

Displaying the Manual

This manual is available in the following formats:

- Portable Document Format (PDF), which can be displayed and printed by using the Adobe Reader, available free from Adobe at **http://www.adobe.com**.
- HTML.

Notational Conventions

A *note box* provides additional information that may further your understanding of a particular topic in this manual:

Note	Note boxes often highlight information that you may need to be aware of when using a particular feature.

A *platform note* provides platform-specific information for a particular procedural step:

Linux	On Linux, you must log on as the `root` user to make many important changes.

A *caution box* provides important information that you should check and understand, prior to starting a particular procedure or reading a particular section of this manual:

Caution!	Be sure to pay attention to these paragraphs because Caution boxes are important!

Typographical Conventions

This manual uses the following typographical conventions:

- User interface components such as icon names, menu names, buttons and selections are displayed in **bold**, for example:

  Click **Next** to continue.

- Commands to be typed are displayed in a `monotype` font, for example:

  At the command prompt, type `admin`.

- File listings and system names (such as file names, directories and database fields) are displayed in a `monotype` font.

Contents

Trademarks

Throughout this manual, *Windows* refers generically to Microsoft Windows 2000, XP, 2003 Server, Vista, 2008 server, 7, 8 or 10, which are trademarks of the Microsoft Corporation.

SQL Server is a registered trademark of the Microsoft Corporation. SQL Azure is a trademark of the Microsoft Corporation.

Note also that although the name UNIX is a registered trademark of The Open Group, the term has come to encompass a whole range of UNIX-like operating systems, including the free, public Linux and even the proprietary Solaris. Easysoft use Unix (note the case) as a general term covering the wide range of Open and proprietary operating systems commonly understood to be Unix ‘flavors’.

Mac OS is a trademark of Apple Computer, Inc., registered in the U.S. and other countries.

Easysoft and Easysoft Data Access are trademarks of Easysoft Limited.

CHAPTER 1 INTRODUCTION

Introducing the Easysoft ODBC-SQL Server Driver

The Easysoft ODBC-SQL Server Driver is an ODBC 3.81 driver for Microsoft SQL Server. It lets ODBC-enabled applications access SQL Server databases from Linux, Unix and Windows platforms.

The Easysoft ODBC-SQL Server Driver supports:

- SQL Server 7.0
- SQL Server 2000
- SQL Server 2005 and SQL Server 2005 Express
- SQL Server 2008 and SQL Server 2008 Express
- SQL Server 2012 and SQL Server 2012 Express
- SQL Server 2014 and SQL Server 2014 Express
- SQL Server 2016 and SQL Server 2016 Express
- SQL Server 2017 and SQL Server 2017 Express
- SQL Server 2019
- SQL Azure

Chapter Guide

Overview

The Easysoft ODBC-SQL Server Driver connects ODBC-enabled applications on Linux, Unix and Windows to SQL Server databases. For example, access SQL Server databases from Apache, ApplixWare, Informatica, Apache OpenOffice, LibreOffice, OpenOffice.org and StarOffice. In addition, the Easysoft ODBC-SQL Server Driver supports the Perl DBI and DBD::ODBC modules, PHP, PEAR DB, the Python pyodbc and mxODBC interfaces, C and any other ODBC-enabled programming language or interface.

The Easysoft ODBC-SQL Server Driver supports SQL Server 7.0 through to 2019. The driver's SQL Server feature support includes database mirroring, encrypted connections using a self-signed SSL certificate, MARS, XML data types, MAX data types, snapshot isolation, GEOGRAPHY and GEOMETRY data types, UTF-16 Supplementary Characters (SC) Collations, contained databases, AlwaysOn Availability Groups and read-only routing.

Product Status

The Easysoft ODBC-SQL Server Driver is currently available on Unix, Linux and Windows platforms. The most up to date list of Easysoft ODBC-SQL Server Driver platforms is available at:

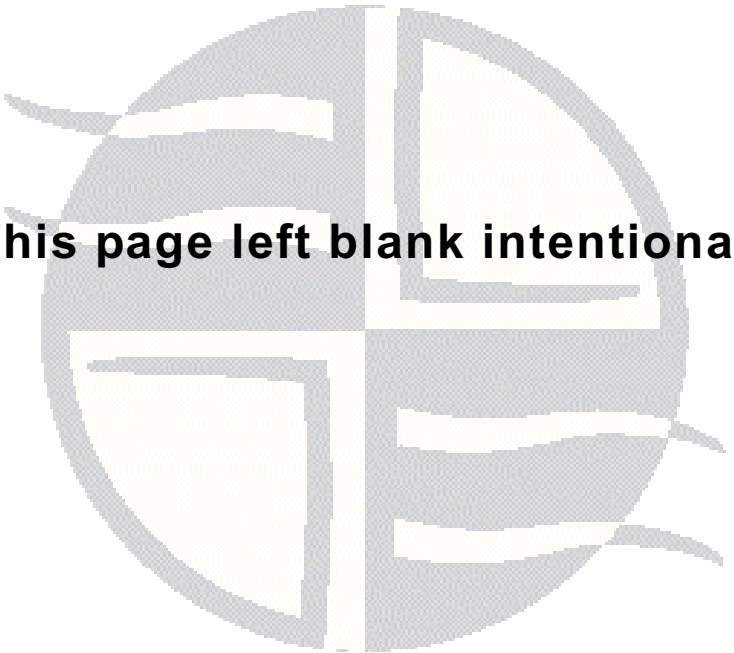
http://www.easysoft.com/products/data_access/odbc-sql-server-driver/index.html

Software problems can be reported to **support@easysoft.com** by users who have either purchased support or registered at the Easysoft web site at **http://www.easysoft.com** and are evaluating Easysoft products.

Deployment

The Easysoft ODBC-SQL Server Driver uses the Tabular Data Stream (TDS) data transfer protocol to communicate with SQL Server. No additional software needs to be installed on the SQL Server machine.

The TCP/IP protocol must be enabled in the SQL Server instance that you want to connect to.



This page left blank intentionally

CHAPTER 2 INSTALLATION

Installing the Easysoft ODBC-SQL Server Driver

This chapter explains how to install, license and remove the Easysoft ODBC-SQL Server Driver.

The Windows installation can be carried out by anyone with local administrator privileges for the target machine.

The Unix installation instructions assume you are, or are able to consult with, a system administrator.

Chapter Guide

Obtaining the Easysoft ODBC-SQL Server Driver

There are two ways to obtain the Easysoft ODBC-SQL Server Driver:

- The Easysoft web site is available 24 hours a day at **http://www.easysoft.com** and lets you download product releases and documentation.

  Choose **Download** from the Easysoft ODBC-SQL Server Driver section of the web site and then choose the platform release that you require.

  If you have not already done so, you will need to register at the web site to download Easysoft software.

- The Easysoft FTP site is available 24 hours a day at **ftp://ftp.easysoft.com** and lets you download free patches, upgrades, documentation and beta releases of Easysoft products, as well as definitive releases.

  Change to the `pub/sql_server` subdirectory and then choose the platform release that you require.

What to Install

The name of the Easysoft ODBC-SQL Server Driver distribution file varies from platform to platform. The file name format is:

- `odbc-sqlserver-`*`x.y.z-platform-variation`*`.tar` (Unix)

– OR –

- `odbc-sql-server-x_y_z-windows.exe` (Windows)

– OR –

- `odbc-sqlserver-`*`x.y.z-platform-variation`*`.dmg` (Mac OS X)

  where:

  `x` is the major version number, *`y`* is the minor version number and *`z`* is the build index, which is incremented when minor changes are made.

  `platform-variation` refers to alternative versions available for a single platform.

Note	Select the highest release available for your platform within your licensed major version number (installing software with a different major version number requires a new Easysoft license).

Unix file names may also be suffixed with `.gz` for a gzipped archive, `.bz2` for a bzip2ed archive, or `.Z` for a compressed archive.

Note

If you download a Unix file with a Windows browser, the browser may strip the file name extension. For example, if you download a `.gz` file and the browser strips the file name extension, it may not be obvious that the file is gzipped. Use `file` *filename* to find out the file type of the downloaded file.

Caution!

As long as you stop all Easysoft software first (or software that uses the Easysoft drivers), it is safe to reinstall or upgrade the Easysoft ODBC-SQL Server Driver without uninstalling.

If you do uninstall, you should first back up any configuration data that you still need, as uninstalling some Easysoft products will result in this information being deleted (license details remain in place).

To continue, refer to the installation instructions for your platform:

- **"Installing the Easysoft ODBC-SQL Server Driver on Unix" on page 21**
- **"Installing the Easysoft ODBC-SQL Server Driver on Mac OS X" on page 58**

Installing the Easysoft ODBC-SQL Server Driver on Unix

These instructions show how to install the Easysoft ODBC-SQL Server Driver on Unix platforms. Please read this section carefully **before** installing the Easysoft ODBC-SQL Server Driver.

BEFORE YOU INSTALL

Requirements

To install the Easysoft ODBC-SQL Server Driver on Unix you need:

- The Bourne shell in `/bin/sh`. If your Bourne shell is not located there, you may need to edit the first line of the installation script.
- Various commonly used Unix commands such as:

  `grep`, `awk`, `test`, `cut`, `ps`, `sed`, `cat`, `wc`, `uname`, `tr`, `find`, `echo`, `sum`, `head`, `tee`, `id`

  If you do not have any of these commands, they can usually be obtained from the **Free Software Foundation**. As the `tee` command does not work correctly on some systems, the distribution includes a `tee` replacement.

- Depending on the platform, you will need up to 10 MB of temporary space for the installation files and up to 10 MB of free disk space for the installed programs. If you also install the unixODBC Driver Manager, these numbers increase by approximately 1.5 MB.
- For Easysoft Licensing to work, you must do one of the following:
  - Install the Easysoft ODBC-SQL Server Driver in `/usr/local/easysoft`.
  - Install the Easysoft ODBC-SQL Server Driver elsewhere and symbolically link `/usr/local/easysoft` to wherever you chose to install the software.

    The installation will do this automatically for you so long as you run the installation as someone with permission to create `/usr/local/easysoft`.
  - Install the Easysoft ODBC-SQL Server Driver elsewhere and set the `EASYSOFT_ROOT` environment variable.

    For more information about setting the `EASYSOFT_ROOT` environment variable, see **"Post installation" on page 44**.
- An ODBC Driver Manager. Easysoft ODBC-SQL Server Driver distributions include the unixODBC Driver Manager.

- You do not have to be the `root` user to install, but you will need permission to create a directory in the chosen installation path. Also, if you are not the `root` user, it may not be possible for the installation to:

  1. Register the Easysoft ODBC-SQL Server Driver with unixODBC.
  2. Create the example data source in the SYSTEM `odbc.ini` file.
  3. Update the dynamic linker entries (some platforms only).

  If you are not `root`, these tasks will have to be done manually later.

  Easysoft recommend you install all components as the `root` user.

What you can Install

This distribution contains:

- The Easysoft ODBC-SQL Server Driver.
- The Easysoft ODBC-SQL Server Driver with SSL Support.
- The unixODBC Driver Manager.

You will need an ODBC Driver Manager to use the Easysoft ODBC-SQL Server Driver from your applications. The distribution therefore contains the **unixODBC Driver Manager**. Most (if not all) Unix applications and interfaces support the unixODBC Driver Manager. For example, Perl DBD::ODBC, PHP, Python and so on.

You do not have to install the unixODBC Driver Manager included with this distribution. You can use an existing copy of unixODBC. For example, a version of unixODBC installed by another Easysoft product, a version obtained from your operating system vendor or one that you built yourself. However, as Easysoft ensure that the unixODBC distributed with the Easysoft ODBC-SQL Server Driver has been tested with that driver, we recommend you use it.

If you choose to use an existing unixODBC Driver Manager, the installation script will attempt to locate it. The installation script looks for the Driver Manager in the standard places. If you have installed it in a non-standard location, the installation script will prompt you for the location. The installation primarily needs unixODBC's `odbcinst` command to install drivers and data sources.

Where to Install

This installation needs a location for the installed files. The default location is `/usr/local`.

At the start of the installation, you will be prompted for an installation path. All files are installed in a subdirectory of your specified path called `easysoft`. For example, if you accept the default location `/usr/local`, the product will be installed in `/usr/local/easysoft` and below.

If you choose a different installation path, the installation script will try to symbolically link `/usr/local/easysoft` to the `easysoft` subdirectory in your chosen location. This allows us to distribute binaries with built in dynamic linker run paths. If you are not `root` or the path `/usr/local/easysoft` already exists and is not a symbolic link, the installation will be unable to create the symbolic link. For information about how to correct this manually, see **"Post Installation Steps for non-root Installations" on page 44**.

Note that you cannot license Easysoft products until either of the following is true:

- `/usr/local/easysoft` exists either as a symbolic link to your chosen installation path or as the installation path itself.
- You have set `EASYSOFT_ROOT` to *`installation_path`*`/easysoft`.

Changes Made to Your System

This installation script installs files in subdirectories of the path requested at the start of the installation, Depending on what is installed, a few changes may be made to your system:

1. If you choose to install the Easysoft ODBC-SQL Server Driver into unixODBC, unixODBC's `odbcinst` command will be run to add an entry to your `odbcinst.ini` file. You can locate this file with `odbcinst -j`. (`odbcinst` is in *`installation_path`*`/easysoft/unixODBC/bin`, if you are using the unixODBC included with this distribution.)

   The Easysoft ODBC-SQL Server Driver distribution includes two drivers, one with SSL Support (`Easysoft ODBC-SQL Server SSL`) and one without SSL support (`Easysoft ODBC-SQL Server`). To access SQL Server over an encrypted connection, you must use the SSL-compatible version of the driver. For more information about the Easysoft ODBC-SQL Server Driver with SSL Support, see **"Encrypting Connections to SQL Server" on page 240**.

   The `odbcinst.ini` entry for the Easysoft ODBC-SQL Server Driver will look similar to this:

```
[Easysoft ODBC-SQL Server]
Driver          = /usr/local/easysoft/sqlserver/lib/libessqlsrv.so
Setup           = /usr/local/easysoft/sqlserver/lib/libessqlsrvS.so
Threading       = 0
FileUsage       = 1
DontDLClose     = 1
UsageCount      = 1
```

The `odbcinst.ini` entry for the Easysoft ODBC-SQL Server Driver with SSL Support will look similar to this:

```
[Easysoft ODBC-SQL Server SSL]
Driver      = /usr/local/easysoft/sqlserver/lib/libessqlsrv_ssl.so
Setup       = /usr/local/easysoft/sqlserver/lib/libessqlsrvS.so
Threading   = 0
FileUsage   = 1
DontDLClose = 1
UsageCount  = 1
```

For information about removing these entries, see **"Uninstalling the Easysoft ODBC-SQL Server Driver on Unix" on page 49**.

2. The installation script installs example data sources into unixODBC. The data sources will be added to your SYSTEM `odbc.ini` file. You can locate your SYSTEM `odbc.ini` file by using `odbcinst -j`. The data source for the standard driver will look similar to this:

```
[SQLSERVER_SAMPLE]
Driver          = Easysoft ODBC-SQL Server
Description     = Easysoft SQL Server ODBC driver
Server          = server.domain.com
Port            = 1422
Database        = northwind
User            = sa
Password        = password
Mars_Connection = No
Logging         = 0
LogFile         =
QuotedId        = Yes
AnsiNPW         = Yes
Language        =
Version7        = No
```

The data source for the Easysoft ODBC-SQL Server Driver with SSL Support will look similar to this:

```
[SQLSERVER_SAMPLE_SSL]
Driver                  = Easysoft ODBC-SQL Server SSL
Description             = Easysoft SQL Server ODBC driver
Server                  = server.domain.com
Port                    = 1433
Database                = northwind
User                    = sa
Password                = password
Mars_Connection         = No
Logging                 = 0
LogFile                 =
QuotedId                = Yes
AnsiNPW                 = Yes
Language                =
Version7                = No
Encrypt                 = Yes
TrustServerCertificate  = No
PrivateKeyFile          =
CertificateFile         =
Entropy                 =
```

For information about removing these data sources, see **"Uninstalling the Easysoft ODBC-SQL Server Driver on Unix" on page 49**.

3. Dynamic Linker.

   On operating systems where the dynamic linker has a file listing locations for shared objects (Linux), the installation script will attempt to add paths under the path you provided at the start of the installation to the end of this list. On Linux, this is usually the file `/etc/ld.so.conf`.

Reinstalling or Installing When You Already Have Other Easysoft Products Installed

Each Easysoft distribution contains common files shared between Easysoft products. These shared objects are placed in *`installation_path`*`/easysoft/lib`. When you run the installation script, the dates and versions of these files will be compared with the same files in the distribution. The files are only updated if the files being installed are newer or have a later version number.

You should ensure that nothing on your system is using Easysoft software before starting an installation. This is because on some platforms, files in use cannot be replaced. If a file cannot be updated, you will see a warning during the installation. All warnings are written to a file called `warnings` in the directory you unpacked the distribution into.

If the installer detects you are upgrading a product, the installer will suggest you delete the product directory to avoid having problems with files in use. An alternative is to rename the specified directory.

If you are upgrading, you will need a new license from Easysoft to use the new driver.

Gathering Information Required During the Installation

During the installation, you will be prompted for various pieces of information. Before installing, you need to find out whether you have unixODBC already installed and where it is installed. The installation script searches standard places like `/usr` and `/usr/local`. However, if you installed the Driver Manager in a non-standard place and you do not install the included unixODBC, you will need to know the location.

If you want to use the installation to test the connection to your SQL Server machine and interactively create an ODBC data source, you will need:

- The host name of the SQL Server machine.
- The name of an instance to connect to or the port that the instance is listening on.
- A valid SQL Server login name and password that can be used to connect to the database you want to access.

INSTALLATION

Unpacking the Distribution

The distribution for Unix platforms is a tar file. There are multiple copies of the same distribution with different levels of compression. You unpack the distribution in one of the following ways.

If the distribution file has been gzipped (`.gz`), use:

```
gunzip odbc-sqlserver-x.y.z-platform.tar.gz
```

If the distribution file has been bzipped (`.bz2`), use:

```
bunzip2 odbc-sqlserver-x.y.z-platform.tar.bz2
```

If the distribution file has been compressed, (`.Z`), use:

```
uncompress odbcsql-server-x.y.z-platform.tar.Z
```

You may have a distribution file which is not compressed at all (`.tar`). To extract the installation files from the tar file, use:

```
tar -xvf odbc-sqlserver-x.y.z-platform.tar
```

This will create a directory with the same name as the tar file (without the `.tar` postfix) containing further archives, checksum files, an installation script and various other installation files.

Change directory into the directory created by unpacking the tar file.

License to Use

The End-User License Agreement is contained in the file `license.txt`. Be sure to understand the terms of the agreement before continuing, as you will be required to accept the license terms at the start of the installation.

Answering Questions During the Installation

Throughout the installation, you will be asked to answer some questions. In each case, the default choice will be displayed in square brackets and you need only press Enter to accept the default. If there are alternative responses, these will be shown in round brackets; to choose one of these, type the response and press Enter.

For example:

```
Do you want to continue? (y/n) [n]:
```

The possible answers to this question are `y` or `n`. The default answer when you type nothing and press Enter is `n`.

Running the Installer

Before you run the installer, make sure you have read **"Installation" on page 30**. If you are considering running the installation as a non `root` user, we suggest you review this carefully as you will have to get a `root` user to manually complete some parts of the installation afterwards. Easysoft recommend installing as the `root` user. (If you are concerned about the changes that will be made to your system, see **"Changes Made to Your System" on page 25**.)

To start the installation, run:

```
./install
```

You will need to:

- Confirm your acceptance of the license agreement by typing "yes" or "no".

  For more information about the license agreement, see **"License to Use" on page 31**.

- Supply the location where the software is to be installed. Easysoft recommend accepting the default installation path.

  For more information, see **"Where to Install" on page 24**.

> **Note** If you are upgrading, you will need a new license from Easysoft.

Locating or Installing unixODBC

Easysoft strongly recommend you use the unixODBC Driver Manager because:

- The installation script is designed to work with unixODBC and can automatically add Easysoft ODBC-SQL Server Driver and data sources during the installation.
- Most ODBC-enabled applications and interfaces support unixODBC. The Easysoft ODBC-SQL Server Driver and any data sources that you add during the installation will be automatically available to your applications and interfaces therefore.
- The unixODBC project is currently led by Easysoft developer Nick Gorham. This means that there is a great deal of experience at Easysoft of unixODBC in general and of supporting the Easysoft ODBC-SQL Server Driver running under unixODBC. It also means that if you find a problem in unixODBC, it is much easier for us to facilitate a fix.

The installation starts by searching for unixODBC. There are two possible outcomes here:

1. If the installation script finds unixODBC, the following message will be output:

   ```
   Found unixODBC under /unixODBC_path
      and it is version n.n.n
   ```

2. If the installation script cannot find unixODBC in the standard places, you will be asked whether you have it installed.

If unixODBC is installed, you need to provide the unixODBC installation path. Usually, the path required is the directory above where `odbcinst` is installed. For example, if `odbcinst` is in `/opt/unixODBC/bin/odbcinst`, the required path is `/opt/unixODBC`.

If unixODBC is not installed, you should install the unixODBC included with this distribution.

If you already have unixODBC installed, you do not have to install the unixODBC included with the distribution, but you might consider doing so if your version is older than the one included.

The unixODBC in the Easysoft ODBC-SQL Server Driver distribution is not built with the default options in unixODBC's configure line.

Option	Description
--prefix=/etc	This means the default SYSTEM `odbc.ini` file where SYSTEM data sources are located will be `/etc/odbc.ini`.
--enable-drivers=no	This means other ODBC drivers that come with unixODBC are not installed.
--enable-iconv=no	This means unixODBC will not look for a libiconv. Warnings about not finding an iconv library were confusing our customers.

Option	Description
--enable-stats=no	Disables unixODBC statistics, which use system semaphores to keep track of used handles. Many systems do not have sufficient semaphore resources to keep track of used handles. In addition, the statistics are only available in the GUI ODBC Administrator.
--enable-readline=no	This disables readline support in `isql`. We disabled this because it ties `isql` to the version of libreadline on the system we build on. We build on as old a version of the operating system as we can for forward compatibility. Many newer Linux systems no longer include the older readline libraries and so enabling readline support makes `isql` unusable on these systems.
--prefix=/usr/local/easysoft/unixODBC	This installs unixODBC into `/usr/local/easysoft/unixODBC`.

Figure 1: Easysoft unixODBC configure line options.

Installing the Easysoft ODBC-SQL Server Driver

The Easysoft ODBC-SQL Server Driver installation script:

- Installs the driver.
- Registers the driver with the unixODBC Driver Manager.

  If the Easysoft ODBC-SQL Server Driver is already registered with unixODBC, a warning will be displayed that lists the drivers unixODBC knows about. If you are installing the Easysoft ODBC-SQL Server Driver into a different directory than it was installed before, you will need to edit your `odbcinst.ini` file after the installation and correct the Driver and Setup paths. unixODBC's `odbcinst` will not update these paths if a driver is already registered.
- Creates an example Easysoft ODBC-SQL Server Driver data source.

  If unixODBC is installed and you registered the Easysoft ODBC-SQL Server Driver with unixODBC, an example data source will be added to your `odbc.ini` file.

  If a data source called "SQLSERVER_SAMPLE" or "SQLSERVER_SAMPLE_SSL" already exists, the existing data source will be displayed and you have the option to replace it.

Licensing

The *`installation_path`*`/easysoft/license/licshell` program lets you obtain or list licenses.

Licenses are stored in the *`installation_path`*`/easysoft/license/licenses` file. After obtaining a license, you should make a backup copy of this file.

The installation script asks you if you want to request an Easysoft ODBC-SQL Server Driver license:

```
Would you like to request a Easysoft ODBC-SQL
Server Driver license now (y/n) [y]:
```

You do not need to obtain a license during the installation, you can run `licshell` after the installation to obtain or view licenses.

If you answer yes, the installation runs the `licshell` script. The process of obtaining a license is best described in the **Licensing Guide**.

To obtain a license automatically, you will need to be connected to the Internet and allow outgoing connections to `license.easysoft.com` on port 8884. If you are not connected to the Internet or do not allow outgoing connections on port 8884, the License Client can create a license request file that you can mail or fax to Easysoft. You can also supply the details to us by telephone.

Start the License Client. The following menu is displayed:

```
[0] exit

[1] view existing license

[n] obtain a license for the desired product.
```

To obtain a license, select one of the options from `[2]` onwards for the product you are installing. The License Client will then run a program that generates a key that is used to identify the product and operating system (we need this key to license you).

After you have chosen the product to license (Easysoft ODBC-SQL Server Driver), you need to supply:

- Your full name.
- Your company name.
- An email contact address. This **must** be the email address that you used when you registered on the Easysoft web site.
- Your telephone number (you need to specify this if you telephone us to request a license).
- Your fax number (you need to specify this if you fax the license request to us).
- A reference number. When applying for a trial license, just press Enter when prompted for a reference number. This field is used to enter a reference number that we will supply you for full (paid) licenses.

You will then be asked to specify how you want to obtain the license. The choices are:

```
[1] Automatically by contacting the Easysoft
License Daemon
```

This requires a connection to the Internet and the ability to support an outgoing TCP/IP connection to `license.easysoft.com` on port 8884.

```
[2] Write information to file so you can fax,
telephone it
```

The license request is output to `license_request.txt`.

```
[3] Cancel this operation
```

If you choose to obtain the license automatically, the License Client will start a TCP/IP connection to `license.easysoft.com` on port 8884 and send the details you supplied and your machine number. No other data is sent. The data sent is transmitted as plain text, so if you want to avoid the possibility of this information being intercepted by someone else on the Internet, you should choose `[2]` and telephone or fax the request to us. The License daemon will return the license key, print it to the screen and make it available to the installation script in the file `licenses.out`.

If you choose option `[2]`, the license request is written to the file `license_request.txt`. You should then exit the License Client by choosing option `[0]` and complete the installation. After you have mailed, faxed or telephoned the license request to us, we will return a license key. Add this to the end of the file *`installation_path`*`/easysoft/license/licenses`.

If any warnings or errors are output during this process, please mail the output to **support@easysoft.com** and we will correct the problem.

Testing the Connection to SQL Server

The Easysoft ODBC-SQL Server Driver installation lets you test the connection to SQL Server, save the connection settings in an ODBC data source and retrieve some SQL Server data. Although the installation default is to do this test, you do not have to.

The installation guides you through the connection process step by step, using `tdshelper` (a diagnostic program supplied with the Easysoft ODBC-SQL Server Driver) to test the SQL Server connection and check that you can access SQL Server with your login name and password. If at any time you want to stop the test, type `q` at any prompt.

If you decide to skip this part of the installation, you can use `tdshelper` after the installation completes to check your SQL Server connection settings. The installation script installs `tdshelper` in the *`installation_path`*`/easysoft/sqlserver/bin` directory.

The installation uses `tdshelper` to search for SQL Server instances that are listening on your network. The results of a successful search will look similar to this:

```
Using /usr/local/easysoft/sqlserver/bin/tdshelper -i -c 1
=============================================================
    ServerName MYSQLSERVER2000HOST              Port 1433 (Default)
    ServerName MYSQLEXPRESSHOST\SQLEXPRESS        Port 2777
    ServerName MYSQLSERVER2005HOST\MYINSTANCEI    Port 1510
    ServerName MYSQLSERVER2005HOST\MYINSTANCEII   Port 1511
=============================================================
```

If you do not see the SQL Server instance that you want to connect to in the list or the list is empty, the SQL Server Browser or SQL Server 2000 listener service may not be running. `tdshelper` uses the SQL Server Browser or listener to find out the available SQL Server instances. If the browser or listener is not running, the installation will be unable to use `tdshelper` to help you interactively connect to SQL Server and create a data source. Type `q` to exit and then manually create a data source after the installation completes. The installation creates a sample data source that you can use as a starting point when setting up your own Easysoft ODBC-SQL Server Driver data sources. For more information about creating data sources, see **"Setting Up Data Sources on Unix" on page 69**.

The example output shows that:

- The default SQL Server instance on a machine named MYSQLSERVER2000HOST is listening on the default SQL Server TCP port 1433.
- The default named SQL Server Express instance on a machine named MYSQLEXPRESSHOST is listening on port 2777.
- There are two named instances running on MYSQLSERVER2005HOST. The instances are listening on ports 1510 and 1511 respectively.

If the SQL Server instance that you want to connect to is listed in the results, type `y` to continue interactively creating your SQL Server data source.

If you chose to continue, type the name (or IP address) of the machine where your SQL Server instance is running when prompted. To connect to a named instance, use the format *machinename**instancename*. To connect to a SQL Server Express instance, use the format *machinename*\SQLEXPRESS. To connect to a SQL Server instance that is not listening on the default port (1433), use the format *machinename:port.*

Based on the example output shown earlier, you would type:

- `MYSQLSERVERHOST` to connect to the default instance on this machine.
- `MYSQLEXPRESSHOST\SQLEXPRESS` to connect to the SQL Server Express instance.
- `MYSQLSERVER2005HOST:1510` to connect to the first named instance on this machine and `MYSQLSERVER2005HOST:1511` to connect to the second.

Type your SQL Server login name when prompted. If you usually connect to SQL Server through your Windows account, type your Windows user name. Use the format *domain**username*, where *domain* is the name of the Windows domain to which *username* belongs.

Otherwise, type a valid SQL Server user name.

Type the password for your user name when prompted.

If `tdshelper` can successfully connect to the SQL Server instance, a list of databases that you can access is displayed.

When setting up your SQL Server login, your database administrator will have associated a database with your login. This is the default database for the connection. The default database is listed first in the `tdshelper` output. If you want to connect to a different database, type the name of another databases in the list. Otherwise, press RETURN to connect to the default database.

If you want to change the language of SQL Server system messages, type one of the listed languages when prompted. Otherwise, press RETURN to accept the default language (again, this is listed first in the `tdshelper` output).

The Easysoft ODBC-SQL Server Driver installation has now gathered enough information to connect to SQL Server. The installation lets you save this connection information in an ODBC data source. You can use this data source to connect to SQL Server now and when the installation completes. The data source is written to your system `odbc.ini` file.

Finally, the installation prompts you whether to retrieve version information from the SQL Server database. The installation uses unixODBC's `isql` and your new data source to do this. Note that if you chose not to license the Easysoft ODBC-SQL Server Driver earlier in the installation, skip this step. The Easysoft ODBC-SQL Server Driver needs to be licensed before it can be used to connect to a data source. When the installation has finished, you can use `isql` to test the data source after you have licensed the Easysoft ODBC-SQL Server Driver.

POST INSTALLATION

Supplied Documents and Examples

The last part of the installation runs a post install script that lists the resources available to you.

- The Easysoft ODBC-SQL Server Driver documentation is installed in *installation_path*/easysoft/sqlserver/doc:
  - The Easysoft ODBC-SQL Server Driver manual in PDF format.
  - The Easysoft ODBC-SQL Server Driver EULA.

installation_path/easysoft/sqlserver/doc/CHANGES.txt lists all the changes in each version of the Easysoft ODBC-SQL Server Driver.

There are also many resources at the **Easysoft web site**.

Post Installation Steps for non-root Installations

If you installed the Easysoft ODBC-SQL Server Driver as a non-root user (not recommended), there may be some additional steps you will need to do manually:

1. If you attempt to install the Easysoft ODBC-SQL Server Driver under the unixODBC Driver Manager and you do not have write permission to unixODBC's odbcinst.ini file, the driver cannot be added.

   You can manually install the driver under unixODBC by adding an entry to the odbcinst.ini file. Run odbcinst -j to find out the location of the DRIVERS file then append the lines from the drv_template file to the odbcinst.ini file. (drv_template is in the directory where the distribution was untarred to).

To manually install the Easysoft ODBC-SQL Server Driver with SSL Support, append the lines from `drv_template_ssl` file to the `odbcinst.ini` file.

2. No example data sources can be added into unixODBC if you do not have write permission to the SYSTEM `odbc.ini` file. Run `odbcinst -j` to find out the location of the `SYSTEM DATA SOURCES` file then add your data sources to this file.

3. On systems where the dynamic linker has a configuration file defining the locations where it looks for shared objects (Linux), you will need to add:

   ```
   installation_path/easysoft/lib
   installation_path/easysoft/unixODBC/lib
   ```

   The latter entry is only required if you installed the unixODBC included with this distribution. Sometimes, after changing the dynamic linker configuration file, you need to run a program to update the dynamic linker cache. (For example, `/sbin/ldconfig` on Linux.)

4. If you did not install the Easysoft ODBC-SQL Server Driver in the default location, you need to do one of the following:

   - Link `/usr/local/easysoft` to the `easysoft` directory in your chosen installation path.

     For example, if you installed in `/home/user`, the installation will create `/home/user/easysoft` and you need to symbolically link `/usr/local/easysoft` to `/home/user/easysoft`:

     ```
     ln -s /home/user/easysoft /usr/local/easysoft
     ```

   - Set and export the `EASYSOFT_ROOT` environment variable to *`installation_path`*`/easysoft`.

5. If your system does not have a dynamic linker configuration file, you need to add the paths listed in step 3 to whatever environment path the dynamic linker uses to locate shared objects. You may want to amend this in a system file run whenever someone logs in such as `/etc/profile`.

   The environment variable depends on the dynamic linker. Refer to your `ld` or `ld.so` man page. It is usually:

   `LD_LIBRARY_PATH`, `LIBPATH`, `LD_RUN_PATH` or `SHLIB_PATH`.

SETTING DYNAMIC LINKER SEARCH PATHS

Your applications will be linked against an ODBC Driver Manager, which will load the ODBC Driver you require. The dynamic linker needs to know where to find the ODBC Driver Manager shared object. The ODBC Driver Manager will load the Easysoft ODBC-SQL Server Driver, which is dependent on further common Easysoft shared objects; the dynamic linker needs to locate these too.

On operating systems where the dynamic linker has a file specifying locations for shared objects (Linux, for example), the installation will attempt to add paths under the path you provided at the start of the installation to the end of this list; no further action should be required. For more information, see **"Dynamic Linker." on page 29**.

On other Unix platforms, there are two methods of telling the dynamic linker where to look for shared objects:

1. You add the search paths to an environment variable and export it.

   This method always works and overrides the second method, described below.

2. At build time, a run path is inserted into the executable or shared objects. On most System V systems, Easysoft distribute Easysoft ODBC-SQL Server Driver shared objects with an embedded run path. The dynamic linker uses the run path to locate Easysoft ODBC-SQL Server Driver shared object dependencies.

   For the first method, the environment variable you need to set depends on the platform (refer to the platform documentation for `ld(1)`, `dlopen` or `ld.so(8)`).

Environment Variable	Platform
LD_LIBRARY_PATH	System V based operating systems and Solaris.
LIBPATH	AIX
SHLIB_PATH	HP-UX
LD_RUN_PATH	Many platforms use this in addition to those listed above.

Figure 2: Dynamic linker search path environment variables.

To use the Easysoft ODBC-SQL Server Driver, you need to add:

installationdir`/easysoft/sqlserver:`*installationdir*`/easysoft/lib`

where *installationdir* is the directory in which you chose to install the Easysoft ODBC-SQL Server Driver. If you accepted the default location, this is `/usr/local`.

An example of setting the environment path in the Bourne shell on Solaris is:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/easysoft/sqlserver:/usr/local/easysoft/lib

export LD_LIBRARY_PATH
```

> **Note** The exact command you need to set and export an environment variable depends on your shell.

If you installed the unixODBC Driver Manager included in the Easysoft ODBC-SQL Server Driver distribution, you also need to add *installationdir*`/easysoft/unixODBC/lib` to the dynamic linker search path.

Uninstalling the Easysoft ODBC-SQL Server Driver on Unix

There is no automated way to remove the Easysoft ODBC-SQL Server Driver in this release. However, removal is quite simple. To do this, follow these instructions.

To uninstall the Easysoft ODBC-SQL Server Driver

1. Change directory to *installation_path*`/easysoft` and delete the `sqlserver` directory. *installation_path* is the Easysoft ODBC-SQL Server Driver installation directory, by default `/usr/local`.
2. If you had to add this path to the dynamic linker search paths (for example, `/etc/ld.so.conf` on Linux), remove it. You may have to run a linker command such as `/sbin/ldconfig` to get the dynamic linker to reread its configuration file. Usually, this step can only be done by the `root` user.
3. If you were using unixODBC, the Easysoft ODBC-SQL Server Driver entry needs to be removed from the `odbcinst.ini` file. To check whether the Easysoft ODBC-SQL Server Driver is configured under unixODBC, use `odbcinst -q -d`. If the command output contains `[Easysoft ODBC-SQL Server Driver]` and `[Easysoft ODBC-SQL Server Driver SSL]`, uninstall the drivers from unixODBC by using:

   ```
   odbcinst -u -d -n 'Easysoft ODBC-SQL Server'

   odbcinst -u -d -n 'Easysoft ODBC-SQL Server SSL'
   ```

   If a reduced usage count message is displayed, repeat this command until `odbcinst` reports that the drivers have been removed.

4. If you created any Easysoft ODBC-SQL Server Driver data sources under unixODBC, you may want to delete these. To do this, first use `odbcinst -j` to locate USER and SYSTEM `odbc.ini` files. Then check those files for data sources that have the driver attribute set to `Easysoft ODBC-SQL Server`.
5. Remove `sqlserver_install.info` from the *`installation_path`*`/easysoft` directory.

Installing the Easysoft ODBC-SQL Server Driver on Windows

Installing the Easysoft ODBC-SQL Server Driver

- Execute the file distribution that you downloaded in **"Obtaining the Easysoft ODBC-SQL Server Driver" on page 18**

  Follow the on screen instructions.

Updating Files That are in Use

To avoid rebooting your computer, the Easysoft ODBC-SQL Server Driver installer prompts you when files that it needs to update are in use by another application or service. This frees the locked files and allows the installation to complete without a system restart.

On Windows Vista and later, the Easysoft ODBC-SQL Server Driver installer uses the Restart Manager to locate the applications that are using files that need updating. These applications are displayed in the Files in Use dialog box. To avoid a system restart, choose **Automatically close applications and attempt to restart them after setup is complete**. The Easysoft ODBC-SQL Server Driver installer then uses the Restart Manager to try to stop and restart each application or service in the list. If possible, the Restart Manager restores applications to the same state and with the same data that they were in before it shut them down.

On earlier versions of Windows, when the Files in Use dialog is displayed, manually shut down each application in the list and then click **Retry** to avoid a system restart.

LICENSING ON WINDOWS

The install program starts the Easysoft License Manager (documented in the **Licensing Guide**), because you cannot use the Easysoft ODBC-SQL Server Driver until a license is obtained.

The following types of license are available:

- a *free time-limited trial license* which gives you free and unrestricted use of the product for a limited period (usually 14 days).
- a *full license* if you have purchased the product. On purchasing the product you are given an authorization code, which you use to obtain a license.

6. Enter your contact details.

   You **MUST** enter the **Name**, **E-Mail Address** and **Company** fields.

   The **Telephone** and **Facsimile** fields are important if you require Easysoft to contact you by those methods.

   The **E-Mail Address MUST** be the same as the address used to register and download from the Easysoft web site or you will be unable to obtain trial licenses.

7. Click **Request License**.

   You are asked for a license type.

8. For a trial license click **Time Limited Trial** and then click **Next**.

   The License Manager asks what software you are licensing:

   Select your required version of the Easysoft ODBC-SQL Server Driver (Standard or Remote, for example) from the drop-down list and then click **Next**.

   – OR –

   If you have obtained an authorization code for a purchased license, select **Non-expiring License** and then click **Next**.

   The License Manager requests your authorization code.

   Enter the authorization code and then click **Next**.

9. The License Manager displays a summary of the information you entered and allows you to choose the method of applying for your license.

10. Choose **On-line Request** if your machine is connected to the internet and can make outgoing connections to port 8884.

The License Manager then sends a request to the Easysoft license server to activate your license key automatically. This is the quickest method and results in your details being entered immediately into our support database. You can now go to **step on page 64**.

NB	Only your license request identifier and contact details as they are displayed in the main License Manager screen are sent to Easysoft.

The remaining three options (**Email Request**, **Print Request** and **View Request**) are all ways to obtain a license if your machine is off-line (i.e. does not have a connection to the internet).

Each of these methods involves providing Easysoft with information including your machine number (a number unique to your machine) and then waiting to receive your license key.

Instead of emailing your details to Easysoft, you can enter them directly at the Easysoft web site and your license key will be emailed to you automatically.

To use this method, click **View Request**, and then visit:

- **http://www.easysoft.com/support/licensing/trial_license.html**

  (trial licenses)

- **http://www.easysoft.com/support/licensing/full_license.html**

  (purchased licenses)

In the Licensing page, enter your machine number (and authorization code for purchased license), click **Submit** and your license key will be emailed to you.

NB	You can copy your machine number from the **View Request** dialog box using `CTRL-C` and then paste it into the License Generator by using `CTRL-V`.

When you receive the license key, you can activate it either by double-clicking the email attachment or by clicking **Enter License** on the License Manager main screen and pasting the license key into the dialog box.

11. A message tells you how many licenses have been added.

> **NB** If you use the **Email Request** option, the license key is emailed to the email address as displayed on the License Manager screen, not the `from:` address of your email.

For more information about the licensing procedure refer to the **Licensing Guide**.

12. Click **Finish** in the License Manager.

The installation is complete.

REPAIRING THE EASYSOFT ODBC-SQL SERVER DRIVER INSTALLATION

The installer can repair a broken Easysoft ODBC-SQL Server Driver installation. For example, you can use the installer to restore missing Easysoft ODBC-SQL Server Driver files or registry keys.

In Windows Vista and later versions of Windows:

1. In **Control Panel**, open **Programs and Features**.
2. Right-click **Easysoft ODBC-SQL Server Driver**, and then click **Repair**.

In earlier versions of Windows:

1. In **Control Panel**, open **Add or Remove Programs**.
2. Select **Easysoft ODBC-SQL Server Driver** and click **Change/Remove**.

Uninstalling the Easysoft ODBC-SQL Server Driver on Windows

This section explains how to remove the Easysoft ODBC-SQL Server Driver from your system.

Removing Easysoft ODBC-SQL Server Driver Data Sources

Easysoft ODBC-SQL Server Driver data sources are not removed when you uninstall. You therefore do not need to recreate your Easysoft ODBC-SQL Server Driver data sources if you reinstall or upgrade. If you do not want to keep your Easysoft ODBC-SQL Server Driver data sources, use ODBC Administrator to remove them, **before** uninstalling the Easysoft ODBC-SQL Server Driver.

1. In **Control Panel**, double-click **Administrative Tools** and then **Data Sources (ODBC)**.
2. Select the data source in the **ODBC Administrator** and click the **Remove** button.

64-bit Windows

There is both a 32-bit and a 64-bit version of ODBC Administrator. The 64-bit ODBC Administrator is located in Control Panel under Administrative tools. To access the 32-bit ODBC Administrator, in the Windows Run dialog box, type:

```
%windir%\syswow64\odbcad32.exe
```

If you do not see the data source in the 64-bit ODBC Administrator, look for it in the 32-bit ODBC Administrator.

REMOVING THE EASYSOFT ODBC-SQL SERVER DRIVER

In Windows Vista and later versions of Windows:

1. In **Control Panel**, open **Programs and Features**.
2. Double-click **Easysoft ODBC-SQL Server Driver**.

In earlier versions of Windows:

1. In **Control Panel**, open **Add or Remove Programs**.
2. Select **Easysoft ODBC-SQL Server Driver** and click **Change/Remove**.

The uninstall process is complete.

Any licenses you obtained for the Easysoft ODBC-SQL Server Driver and other Easysoft products are held in the Windows registry.

When you uninstall, your licenses are not removed so you do not need to relicense the product if you reinstall or upgrade.

Installing the Easysoft ODBC-SQL Server Driver on Mac OS X

This section shows how to install the Easysoft ODBC-SQL Server Driver for Mac OS X.

The Easysoft ODBC-SQL Server Driver for Mac OS X distribution file is a Mac Installer package. The package is contained in a disk image file.

The Easysoft ODBC-SQL Server Driver installer tries to:

- Install the Easysoft ODBC-SQL Server Driver into the iODBC Driver Manager.

  To do this, the installer adds an entry to the `odbcinst.ini` file similar to the following:

```
[ODBC Drivers]
Easysoft SQL Server ODBC Driver = Installed

[Easysoft SQL Server ODBC Driver]
Driver = /usr/local/easysoft/sqlserver/lib/libessqlsrv.so
Setup =
/usr/local/easysoft/sqlserver/lib/libessqlsrvS.bundle/Contents/MacOS/libessqlsrvS
```

  (To locate the `odbcinst.ini` file file, type `iodbc-config --odbcinstini` in a Terminal window.)

For the installer program to install the Easysoft ODBC-SQL Server Driver into the iODBC Driver Manager and create the example data source, the ODBC package included with Mac OS X must already be installed. If your version of Mac OS X does not include the ODBC package, iODBC Driver Manager packages for Mac OS X can be downloaded from the iODBC web site (**http://www.iodbc.org/dataspace/iodbc/wiki/iODBC/Downloads**)

The installer program installs the Easysoft ODBC-SQL Server Driver into `/usr/local/easysoft/sqlserver` directory.

To install the Easysoft ODBC-SQL Server Driver for Mac OS X

1. Log in as an administrator user.
2. Double-click the Easysoft ODBC-SQL Server Driver for Mac OS X disk image file that you obtained from one of the sources described in **"Obtaining the Easysoft ODBC-SQL Server Driver" on page 18**.
3. In the open disk image, double-click the package file to start the Easysoft ODBC-SQL Server Driver installer.

   The Easysoft ODBC-SQL Server Driver installer welcome screen is displayed.
4. When you have read the introductory text, click **Continue**.

   Follow the on screen instructions.

LICENSING THE EASYSOFT ODBC-SQL SERVER DRIVER

You cannot use the Easysoft ODBC-SQL Server Driver until you have obtained a license. The Easysoft License Manager allows you to obtain (and view) licenses.

Licenses are stored in the *installation_path*`/easysoft/license/licenses` file. After obtaining a license, you should make a backup copy of this file. *installation_path* is the Easysoft ODBC-SQL Server Driver installation directory, by default `/usr/local`.

The following types of license are available:

- A *free trial license*, which gives you free and unrestricted use of the product for a limited period (usually 14 days).
- A *full license* if you have purchased the product. On purchasing the product you are given an authorization code, which you use to obtain a license.

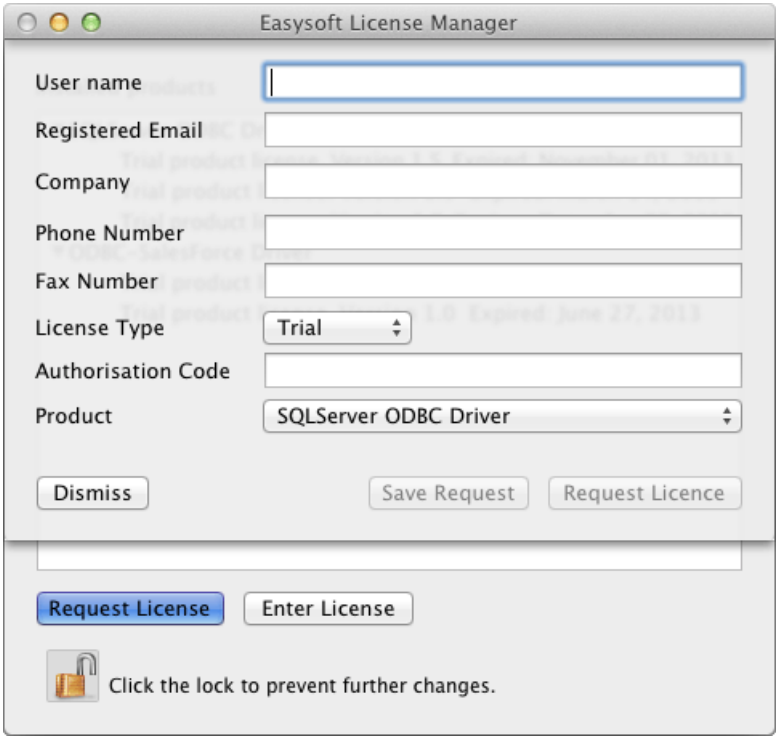


Figure 3: The License Manager window.

To obtain a license automatically, you will need to be connected to the Internet and allow outgoing connections to `license.easysoft.com` on port 8884. If you are not connected to the Internet or do not allow outgoing connections on port 8884, the Easysoft License Manager can create a license request file that you can mail or fax to Easysoft. You can also supply the details to us by telephone.

1. Start the Easysoft License Manager.

   The Easysoft License Manager is located in the `/Applications/Utilities` folder.

2. Choose **Request License**.

   If the Easysoft License Manager window is locked, the **Request License** button will be unavailable. To unlock the Easysoft License Manager window, click the lock icon, and then type an administrator user name and password when prompted.

3. Enter your contact details.

   You MUST complete the **User name**, **Registered E-Mail** and **Company** fields. The e-mail address that you enter here must be the same as the one that you registered with.

   The **Phone Number** and **Fax Number** fields are important if you require Easysoft to contact you by those methods.

4. For a trial license choose **Trial License**.

   – OR –

   If you have obtained an authorisation code for a purchased license, choose **Full**.

5. If you chose **Full** in the previous step, enter your authorisation code in the space provided. Otherwise, skip this step.

6. Choose your required version of the Easysoft ODBC-SQL Server Driver from the drop-down list.

7. Choose **Request License** if your machine is connected to the internet and can make outgoing connections on port 8884.

The License Manager then sends a request to the Easysoft license server to activate your license key automatically. This is the quickest method and results in your details being entered immediately into our support database.

> **Note** Only your license request identifier and contact details as they are displayed in the main License Manager screen are sent to Easysoft.

The remaining option (**Save Request**) lets you obtain a license if your machine is offline (i.e. does not have a connection to the internet).

This method involves providing Easysoft with information including your machine number (a number unique to your machine) and then waiting to receive your license key.

Instead of emailing, faxing or telephoning your details to Easysoft, you can enter them directly at the Easysoft web site and your license key will be emailed to you automatically.

To use this method, choose **Save Request**, and then visit:

- **http://www.easysoft.com/support/licensing/trial_license.html** (trial licenses)
- **http://www.easysoft.com/support/licensing/full_license.html** (purchased licenses)

In the Licensing page, enter your machine number (and authorisation code for purchased license), click **Submit** and your license key will be emailed to you

When you receive the license key, choose **Enter License** on the License Manager main screen and paste the license key into the dialog box.

A message tells you how many licenses have been added.

For more information about the licensing procedure refer to the **Licensing Guide**.

TESTING THE EASYSOFT ODBC-SQL SERVER DRIVER

The Easysoft ODBC-SQL Server Driver only becomes functional when an application is linked with it through an ODBC driver manager. You can use iODBC's command line `iodbctest` application to test the Easysoft ODBC-SQL Server Driver and execute SQL. For more information about using `iodbctest` to test the Easysoft ODBC-SQL Server Driver, see **"Testing Data Sources" on page 85**.

DOCUMENTATION AND RESOURCES

The Easysoft ODBC-SQL Server Driver documentation is installed in the `/usr/local/easysoft/sqlserver/doc` directory. This directory also contains the Easysoft ODBC-SQL Server Driver EULA and change log.

There are also many resources at the **Easysoft web site**.

Uninstalling the Easysoft ODBC-SQL Server Driver on Mac OS X

There is no automated way to remove the Easysoft ODBC-SQL Server Driver on Mac OS X. However, if you want to remove the software manually, Mac OS X provides a way of listing of the files installed by the Easysoft ODBC-SQL Server Driver package. To do this, use the `lsbom` command to examine the Easysoft ODBC-SQL Server Driver package's Bill of Materials (.bom) file.

> **Note** You do not need to remove the Easysoft ODBC-SQL Server Driver before upgrading to a different version.

To list the files installed by the Easysoft ODBC-SQL Server Driver package

- In Terminal, type:

  ```
  lsbom -s -f sqlserver_bomfile
  ```

  where *`sqlserver_bomfile`* is the Easysoft ODBC-SQL Server Driver .bom file. For example:

  ```
  /private/var/db/receipts/com.easysoft.easysoftSqlServerOdbcDriver.package.pkg.bom
  ```

To remove the ODBC driver entry from odbcinst.ini

1. Log in as an administrator user.
2. Open ODBC Administrator in the `/Applications/Utilities` folder.

   If ODBC Administrator is not included with your version of Mac OS X, you can download it from the Apple Support web site (**http://support.apple.com/kb/DL895**).

3. To remove the Easysoft ODBC-SQL Server Driver entry from `odbcinst.ini`, in the **Drivers** tab, click Easysoft ODBC-SQL Server Driver in the list of drivers. Click **Remove.** Click **OK** when prompted.

   If the Drivers tab is locked, the Remove button will be unavailable. To unlock the Drivers tab, click the lock icon, and then type an administrator password when prompted.

4. If you want to remove your Easysoft ODBC-SQL Server Driver data sources, click **OK** when prompted.

5. Click **Apply** to save your changes.

CHAPTER 3 CONFIGURATION

Configuring the Easysoft ODBC-SQL Server Driver

The Easysoft ODBC-SQL Server Driver is installed on the computer where your applications are running. ODBC applications access ODBC drivers through the ODBC Driver Manager and a data source. The data source tells the Driver Manager which ODBC driver to load, which SQL Server instance to connect to and how to connect to it. This chapter describes how to create data sources, use DSN-less connections and configure the Easysoft ODBC-SQL Server Driver.

Before setting up a data source, you must have successfully installed the Easysoft ODBC-SQL Server Driver.

For Easysoft ODBC-SQL Server Driver installation instructions, see **"Installation" on page 17**.

Chapter Guide

Configuring the Easysoft ODBC-SQL Server Driver

This section describes how to configure the Easysoft ODBC-SQL Server Driver to connect to a SQL Server database by using a data source or a DSN-less connection string. The section assumes you are, or are able to consult with, a database administrator.

Setting Up Data Sources on Unix

There are two ways to set up a data source to your SQL Server data:

- Create a SYSTEM data source, which is available to anyone who logs on to this Unix machine.

– OR –

- Create a USER data source, which is only available to the user who is currently logged on to this Unix machine.

By default, the Easysoft ODBC-SQL Server Driver installation creates a SYSTEM data source named `[SQLSERVER_SAMPLE]`. If you are using the unixODBC included in the Easysoft ODBC-SQL Server Driver distribution, the SYSTEM `odbc.ini` file is in `/etc`.

If you built unixODBC yourself, or installed it from some other source, SYSTEM data sources are stored in the path specified with the configure option `--sysconfdir=`*`directory`*. If `sysconfdir` was not specified when unixODBC was configured and built, it defaults to `/usr/local/etc`.

If you accepted the default choices when installing the Easysoft ODBC-SQL Server Driver, USER data sources must be created and edited in `$HOME/.odbc.ini`.

Note

To display the directory where unixODBC stores SYSTEM and USER data sources, type `odbcinst -j`.

By default, you must be logged in as `root` to edit a SYSTEM data source defined in `/etc/odbc.ini`.

You can either edit the sample data source or create new data sources.

Each section of the `odbc.ini` file starts with a data source name in square brackets [ ] followed by a number of *attribute=value* pairs.

> **Note** Attribute names in `odbc.ini` are not case sensitive.

The `Driver` attribute identifies the ODBC driver in the `odbcinst.ini` file to use for a data source. The Easysoft ODBC-SQL Server Driver distribution includes two drivers:

- One with SSL support that should be used if you need to access SQL Server 2000 or later over an encrypted connection.
- One without SSL support that should be used for SQL Server 2000 and later when encryption is not required and for SQL Server 7.0.

When the Easysoft ODBC-SQL Server Driver is installed into unixODBC, entries for the standard driver (`Easysoft ODBC-SQL Server`) and the driver with SSL support (`Easysoft ODBC-SQL Server SSL`) are placed in `odbcinst.ini`.

For Easysoft ODBC-SQL Server Driver data sources, you need to include a `Driver = Easysoft ODBC-SQL Server` entry.

For Easysoft ODBC-SQL Server Driver with SSL Support data sources, you need to include a `Driver = Easysoft ODBC-SQL Server SSL` entry. For more information about configuring Easysoft ODBC-SQL Server Driver with SSL Support data sources, see **"Encrypting Connections to SQL Server" on page 240**.

To configure a SQL Server data source, in your `odbc.ini` file, you need to specify:

- The host name or IP address of the machine where the SQL Server instance is running. To connect to a named instance you also need to specify the instance name. (`Server`)
- A valid SQL Server login name (`User`) and password (`Password`).

For example:

```
[SQL Server]
Driver    = Easysoft ODBC-SQL Server
# To connect to the default instance, omit \my_instance_name.
Server    = my_sqlserver_hostname\my_instance_name
User      = my_domain\my_domain_user
Password  = my_password
```

If the SQL Server Browser or listener service is not in use at your site and you want to connect to an instance that is not listening on the default TCP port (1433), you also need to specify the port: For example, to connect to a SQL Server instance that is listening on port 1500, add this entry:

```
Port              = 1500
```

Setting Up Data Sources on Windows

To connect an ODBC application on a Windows machine to a SQL Server database:

1. Open ODBC Data Source Administrator:

   - For Microsoft Windows 10 and later, use the Windows search facilty to look for "ODBC".
   - For Microsoft Windows 8, in the Windows desktop, point to the upper-right corner of the screen, move the mouse pointer down, and then choose **Settings** > **Control Panel** > **Administrative Tools** > **ODBC Data Sources (64-bit)**.
   - For Microsoft Windows Vista and Windows 7, choose **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.
   - For Microsoft Windows Server 2008 and Windows Server 2008 R2, choose **Start** > **Administrative Tools** > **Data Sources (ODBC)**.
   - For Microsoft Windows 2000, Windows XP and Windows Server 2003, choose **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources**.

   The **ODBC Data Source Administrator** dialog box is displayed.

2. Select the **User DSN** tab to set up a data source that only you can access.

   – OR –

   Select the **System DSN** tab to create a data source which is available to anyone who logs on to this Windows machine.

3. Click **Add...** to add a new data source.

The **Create New Data Source** dialog box displays a list of drivers.

4. Choose the **Easysoft ODBC-SQL Server** Driver and choose **OK**.

   The DSN Setup dialog box is displayed.

   For details of the attributes that can be set on this dialog box, see **"Attribute Fields" on page 86**.

64-bit Windows

The Easysoft installer program installs both a 32-bit and a 64-bit version of the Easysoft ODBC-SQL Server Driver. If you want to use a 64-bit ODBC application, you need to use the 64-bit Easysoft ODBC-SQL Server Driver. If you want to use a 32-bit ODBC application, you need to use the 32-bit Easysoft ODBC-SQL Server Driver.

There is both a 32-bit and a 64-bit version of ODBC Administrator. The 64-bit ODBC Administrator is located in Control Panel under Administrative tools. To access the 32-bit ODBC Administrator in Windows 7 and earlier, in the Windows Run dialog box, type:

```
%windir%\syswow64\odbcad32.exe
```

On Windows 8 and later, both the 32-bit and 64-bit ODBC Administrator are located in Control Panel under Administrative tools: ODBC Data Sources (32-bit) and ODBC Data Sources (64-bit).

Easysoft ODBC-SQL Server Driver data sources created in the 64-bit ODBC Administrator will specify the 64-bit version of the Easysoft ODBC-SQL Server Driver. Easysoft ODBC-SQL Server Driver data sources created in the 32-bit ODBC Administrator will specify the 32-bit version of the Easysoft ODBC-SQL Server Driver.

If you want to create an Easysoft ODBC-SQL Server Driver System data source for use with a 64-bit application, use the 64-bit ODBC Administrator. If you want to create an Easysoft ODBC-SQL Server Driver System data source for use with a 32-bit application, use the 32-bit ODBC Administrator.

For Easysoft ODBC-SQL Server Driver User data sources, it does not matter which version of the ODBC Administrator that you use.

Setting Up Data Sources on Mac OS X

To create an Easysoft ODBC-SQL Server Driver data source on Mac OS X, use ODBC Administrator. ODBC Administrator is a component of iODBC that provides a GUI with which you can add, modify, delete and examine data sources. If ODBC Administrator is not included with your version of Mac OS X, you can download it from the Apple Support web site (**http://support.apple.com/kb/DL895**).

Data sources are stored in the `odbc.ini` file. User data sources are stored in `~/Library/ODBC`. To find out the directory where System data sources are stored, in a Terminal window, type the following command:

```
iodbc-config --odbcini
```

iodbc-config is a script that outputs iODBC configuration information. The `--odbcini` option prints the system wide `odbc.ini` file path. The following line shows some example output from the previous command:

```
/Library/ODBC/odbc.ini
```

Note	To add, remove or edit System data sources, you need to be logged in as an administrator user. Depending on the permissions on the system wide `odbc.ini` file, you may have to log in as the root user to add, remove or edit System data sources. Otherwise, the changes that you make will not be saved in the `odbc.ini` file. By default, the root user account is not active. For information about enabling the root user and the implications of using the root account, in the Mac OS X Help, search for "root user."

To add an Easysoft ODBC-SQL Server Driver data source

1. Open ODBC Administrator in the `/Applications/Utilities` folder.

   ODBC Administrator is displayed.

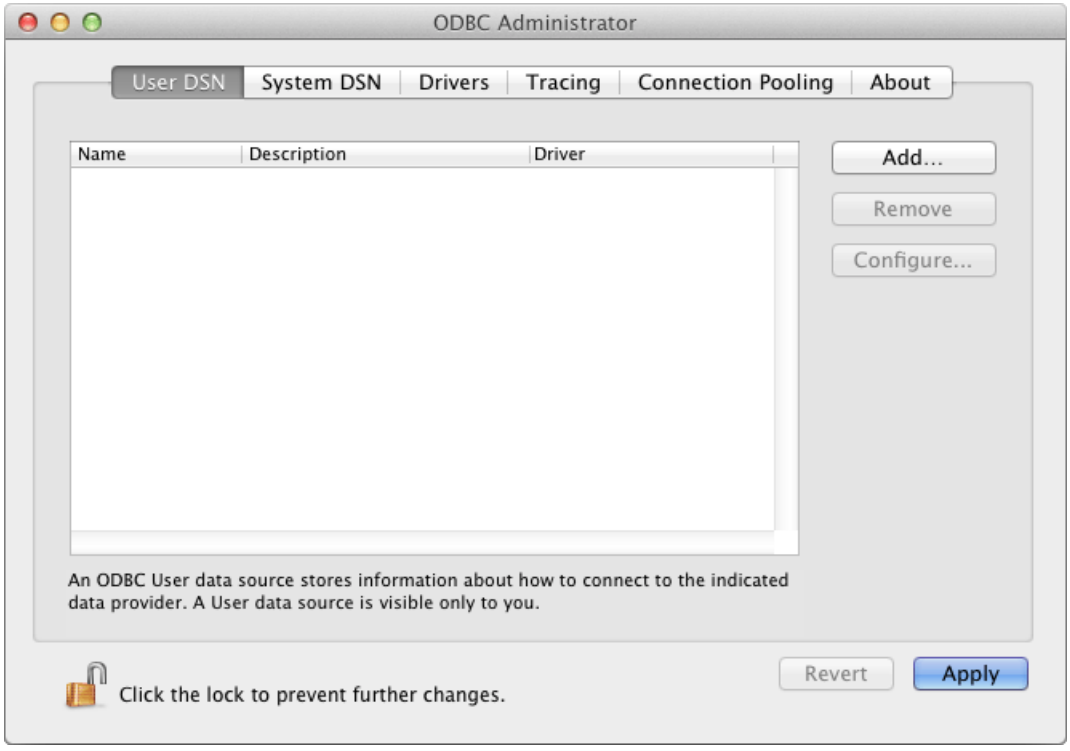


Figure 4: The Mac OS X ODBC Administrator

2. Do one of the following:

   - To add a data source that is only visible to the user currently logged in to this computer, in the **User DSN** tab, click **Add**.
   - To add a data source that is visible to all users of this computer, click the **System DSN** tab, and then click **Add**.

   If the System DSN tab is locked, the Add button will be unavailable. To unlock the System DSN tab, click the lock icon, and then type an administrator user name and password when prompted.

The Choose A Driver dialog prompts you to choose the driver for which you want to set up the data source.

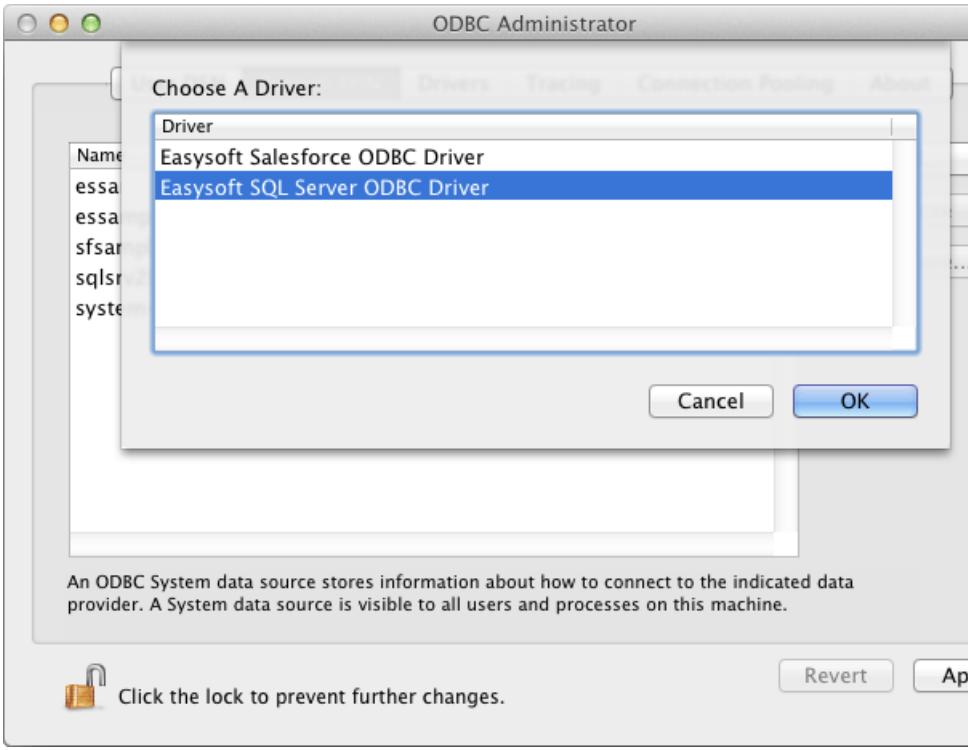


Figure 5: The Mac OS X ODBC Administrator Choose A Driver dialog box

3. From the list of ODBC drivers, choose Easysoft ODBC-SQL Server Driver, then click **OK**.

   The Easysoft ODBC-SQL Server Driver for Mac OS X DSN dialog box is displayed.

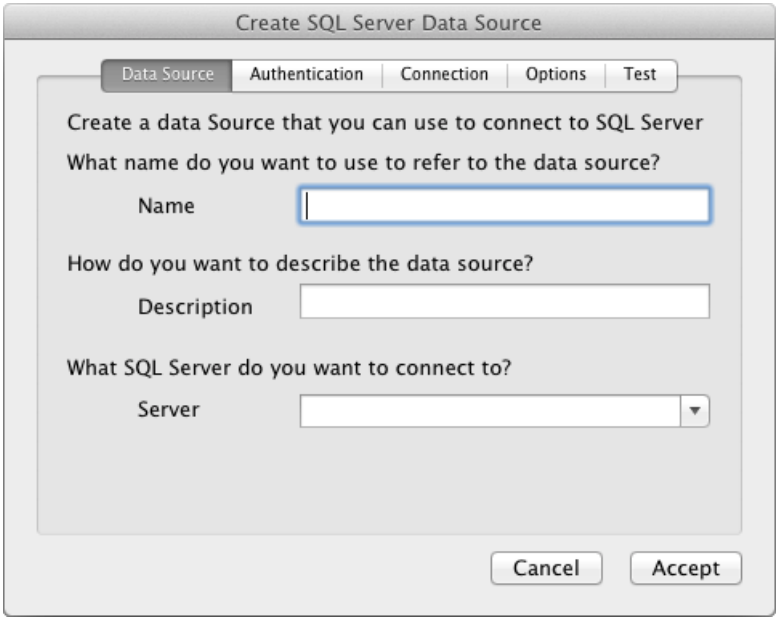


Figure 6: The Create SQL Server Data Source dialog box

4. In the Data Source tab, in the **Name** box, type the data source name.
5. In the **Description** box, type some text to describe the data source.

   Some applications display the description text to help users differentiate between different data sources.
6. In the **Server** box, type the host name or IP address of the machine where the SQL Server instance is running. To connect to a named instance you also need to specify the instance name. For example:

   ```
   my_sqlserver_hostname\my_instance_name
   ```

   Alternatively, choose a SQL Server instance from the drop-down list.
7. Choose the Authentication tab.

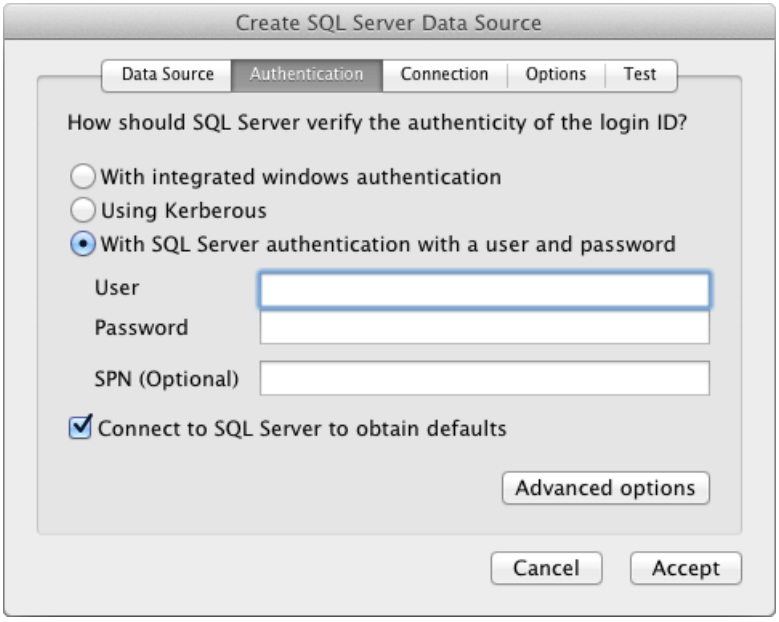


Figure 7: The Create SQL Server Data Source dialog box—Authentication tab

8. Do one of the following:
   - If you want to use a Windows user name to authenticate the connection, choose **With integrated windows authentication**.
   - If you want to use a SQL Server user name to authenticate the connection, choose **With SQL Server authentication with a user and password**.
9. In the **User** box and **Password** boxes, type a valid SQL Server login name and password.

   If the SQL Server login is a Windows user name, use the format:

   domain\username

where:

- *domain* is the name of the Windows domain that the SQL Server machine is in or one that the SQL Server machine trusts.
- *username* is the user name of a user who belongs to this domain.

10. If the SQL Server Browser or listener service is not in use at your site and you want to connect to an instance that is not listening on the default TCP port (1433), you also need to specify the port. To do this choose **Advanced options**, and then type the Port in the **Port** box. Choose **Finished**.

    For example, to connect to a SQL Server instance that is listening on port 1500, type `1500` in the **Port** box.

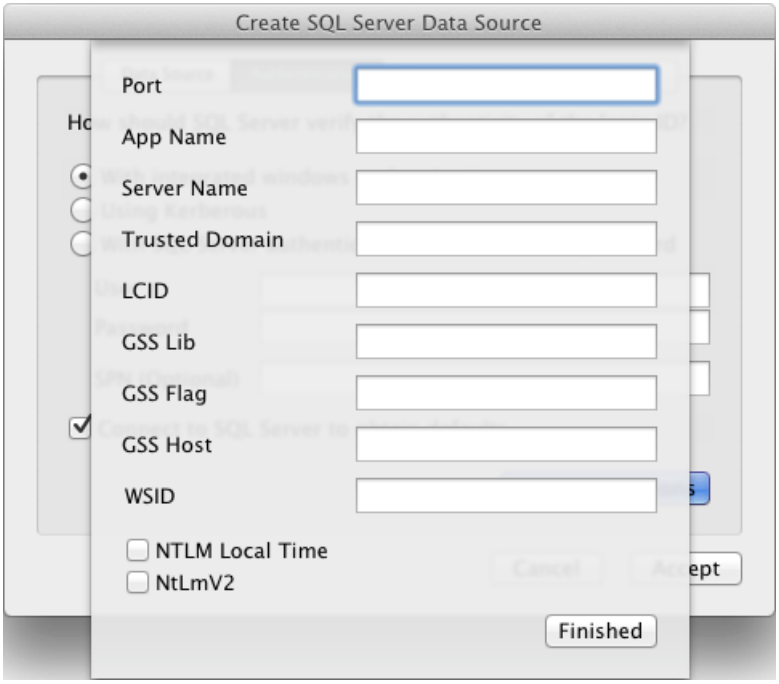


Figure 8: The Create SQL Server Data Source dialog box— Advanced Authentication Options

11. Choose the Connection tab.

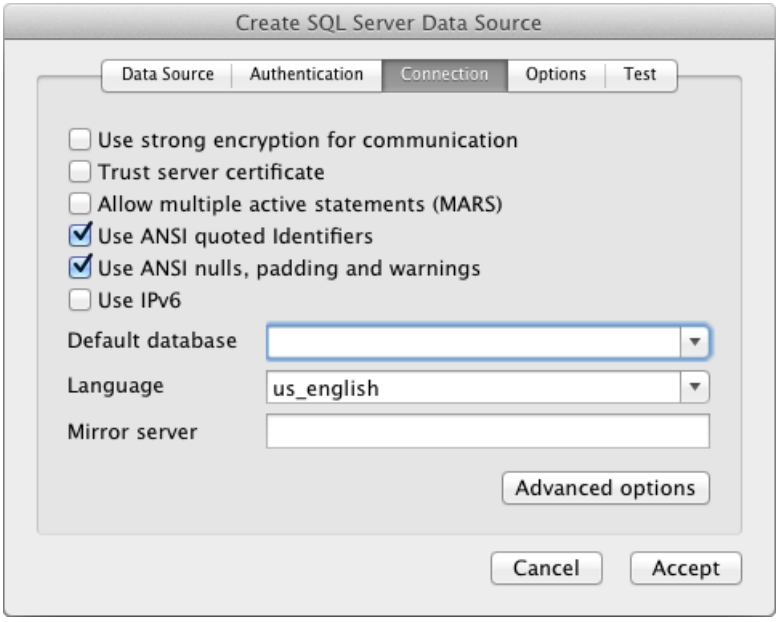


Figure 9: The Create SQL Server Data Source dialog box—Connection tab

12. If you want to change the default database to use for the connection, choose the database you want from the **Default database** list. Note that if the login does not have permission to access the database, the connection will fail.
13. Choose the Test tab.
14. Choose the **Test Connection** button to verify that you can successfully connect to SQL Server with your new data source.

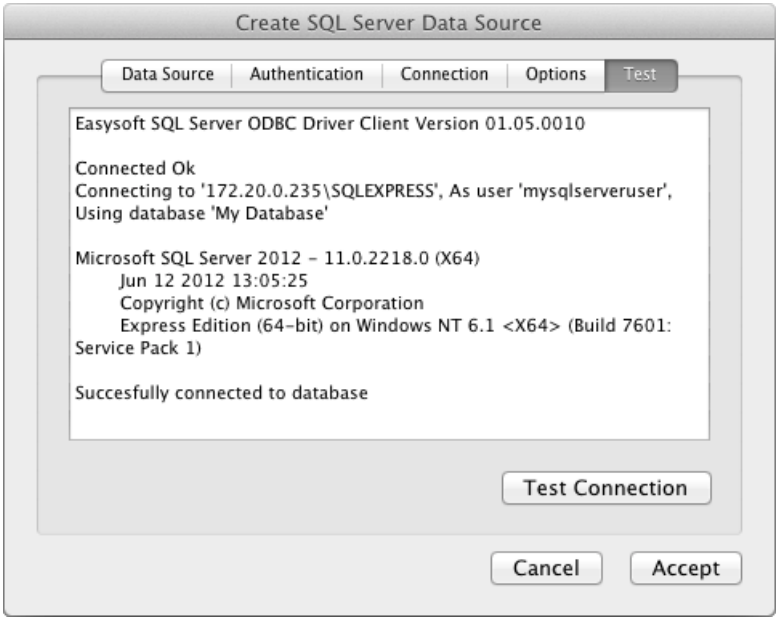


Figure 10: The Create SQL Server Data Source dialog box—Test tab

If the connection fails, refer to **Troubleshooting Database Connection Problems**.

15. In ODBC Administrator, click **Apply** to save your new data source.

TESTING DATA SOURCES

`iodbctest` is a command line ODBC application that is supplied with iODBC. Use it to test your Easysoft ODBC-SQL Server Driver data sources. If a connection can successfully be made, you can query the remote database by executing SQL statements in `iodbctest`.

To test an Easysoft ODBC-SQL Server Driver data source

1. Open Terminal in the `/Applications/Utilities` folder.
2. At the shell prompt, type:

   ```
   iodbctest
   ```

3. Type DSN=*`name`*

   where *`name`* is the name of an Easysoft ODBC-SQL Server Driver data source that you want to test.

   If the connection succeeds, `iodbctest` prompts you to type some SQL.

   If the connection fails, `iodbctest` displays an error to help you identify what the problem is.

4. To exit `iodbctest`, type `quit`.

Attribute Fields

The following Easysoft ODBC-SQL Server Driver attributes may be set in the `odbc.ini` file or the driver's DSN setup dialog box on Windows:

Attribute	Description
Driver = *value*	The name of the ODBC driver to use with this data source. To connect to a SQL Server 2000 or later instance over an encrypted connection, set this attribute value to `Easysoft ODBC-SQL Server SSL`. Otherwise, set this attribute value to `Easysoft ODBC-SQL Server`.
Description = *value*	A single line of descriptive text that may be retrieved by some applications to describe the data source.

Attribute	Description

Server = *value*	The SQL Server instance that you want to connect to. To connect to the default SQL Server instance, type:
	machinename
	where *machinename* is the name or IP address of the host where SQL Server is running.
	Note that if you are connecting to a SQL Server 2005 or later instance that is listening on an IPv6 address, set the `IPv6` attribute to 1.
	To connect to a named instance, type:
	*machinename**instancename*
	where *instancename* is the SQL Server instance.
	To connect to the default SQL Server Express named instance, type:
	machinename\sqlexpress
	On Windows, you also have the option of using a named pipe connection. To use named pipes, use this format for the `Server` value:
	`\\localhost\pipe\MSSQL$`*`instance`*`\sql\query`
	For example:
	`\\localhost\pipe\MSSQL$sqlexpress\sql\query`

Attribute	Description

	Connection Failover
	If your SQL Server database is available on more than one SQL Server machine, you can define a primary server for the database and additional fallback database servers. By default, the Easysoft ODBC-SQL Server Driver will try to connect to the first server that you specify. If that server is unavailable the Easysoft ODBC-SQL Server Driver will try to connect to the next server in the list and so on. Use the format:
	`Server = primaryserver[:port] [, fallbackserver[:port]...]`
	where:
	• *primaryserver* is the name or IP address of the primary SQL Server machine on which your database is available.
	• *port* is the TCP port on which the instance is listening. If omitted, the driver will try to connect to the instance that is listening on port 1433.
	• *fallbackserver* is the name or IP address of an alternative SQL Server machine on which your database is available.
	For example:
	`Server = sqlsrvhostA,sqlsrvhostB,sqlsrvhostC:1583`
	Connection attempts continue until either a connection is successfully made or all the servers in the list have been tried once.
	Note that your SQL Server login (as specified by `User` and `Password`) needs to be valid on each SQL Server machine in the list. The SQL Server login must have permission to access the database on each SQL Server machine.
	If you want to balance the load between database servers, configure the driver to randomly choose the database server it connects to. To do this, set the `ClientLB` attribute to 1.

Attribute	Description
Port = *num*	The TCP port that SQL Server is listening on. If you are connecting to a default instance that is listening on port 1433, the `Port` setting can be omitted. If the SQL Server Browser or the SQL Server 2000 listener service is running, the Easysoft ODBC-SQL Server Driver will automatically detect the port number and the `Port` setting can be omitted. By default, named instances of SQL Server use dynamic ports, which means that an available port is assigned when the instance starts. If a SQL Server instance is listening on a dynamically allocated port number, you must omit the `Port` setting and let the Easysoft ODBC-SQL Server Driver use the browser or listener to detect the port number. If the SQL Server Browser or listener is not running at your site, your database administrator will have configured each SQL Server instance to listen on a specific TCP port. You need to specify this port with the `Port` setting. If your database administrator has hidden the SQL Server instance from the SQL Server Browser or listener, you need to specify the port number of the hidden instance. If your database administrator has configured the SQL Server instance to listen on multiple ports, use the `Port` setting to specify the appropriate port number from the available alternatives.

Attribute	Description
User = *value*	The SQL Server login name to use when connecting to SQL Server. If the SQL Server instance uses Windows Authentication (also known as trusted connections), the Windows user name to use to authenticate the connection. Use this format: *domain\username* where: • *domain* is the name of the Windows domain that the SQL Server machine is in or one that the SQL Server machine trusts. • *username* is the user name of a user who belongs to this domain. If the SQL Server instance permits SQL Server Authentication, you can also specify a SQL Server user name. To specify the login name in the connection string, use `UID` rather than `User`. For more information about specifying Easysoft ODBC-SQL Server Driver attributes in the connection string, see **"DSN-less Connections" on page 128**. **Windows** If you want to access SQL Server with a Windows log in, choose With Integrated Windows Authentication.
Password = *value*	The password for the login name specified by `User`. To specify the login password in the connection string, use `PWD` rather than `Password`.
Authentication = *value*	Controls the authentication mode used by the Easysoft ODBC-SQL Server Driver. Currently, the only value for this attribute is `ActiveDirectoryPassword`, which you need to set if you are connecting to SQL Azure with an Azure Active Directory account user name and password.

Attribute	Description

ServerName = *value*	This attribute is only relevant if you are using SQL Azure and lets you specify the fully qualified domain name (FQDN) of the SQL Azure server that you want to connect to. For example:
	`xyz12345yzx.database.windows.net`
	There are two ways to specify the SQL Azure server, as shown by the following data source extracts:
	`ServerName = xyz12345yzx.database.windows.net`
	`User = myuser`
	– OR –
	`Server = xyz12345yzx.database.windows.net`
	`User = myuser@xyz12345yzx`
	For more information about SQL Azure, see the following Easysoft tutorial:
	http://www.easysoft.com/products/data_access/odbc-sql-azure-driver/linux-unix.html

Database = *value*	The default database to use for the connection. If you omit this attribute, the connection uses the default database defined for the login in SQL Server. The default database for users who do not have their own SQL Server login depends on the local group on the SQL Server machine that they belong to. The default database for members of the local Administrators group is the one defined for the BUILTIN\Administrators login. The default database for members of the local Users group is the one defined for the BUILTIN\Users login (SQL Server Express Edition only). If the database does not exist or the login does not have permission to access the database, the connection will fail. Note that using the default database for the login ID is more efficient than specifying a default database in the ODBC data source.

Attribute	Description

QuotedId = Yes | No	When ON (set to `Yes`), `QUOTED_IDENTIFIERS` is set to ON for the connection. SQL Server will then follow the SQL-92 rules regarding the use of quotation marks in SQL Statements. Double quotes can only be used for identifiers, such as column and table names. Character strings must be enclosed in single quotes:
	`SELECT CompanyName`
	`FROM "Customer and Suppliers by City"`
	`WHERE City = 'New York'`
	If a single quotation mark is part of the literal string, it can be represented by two single quotation marks.
	When OFF, `QUOTED_IDENTIFIERS` is set to OFF for the connection. SQL Server then follows the legacy Transact-SQL rules regarding the use of quotation marks. Identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers. Literals can be delimited by either single or double quotation marks.
	For more information about the `QUOTED_IDENTIFIERS` option, see the SQL Server Transact-SQL documentation.
	By default, `QuotedId` is ON.

Attribute	Description

AnsiNPW = Yes | No	When ON (set to `Yes`), the `ANSI_NULLS`, `ANSI_WARNINGS`, and `ANSI_PADDING` options are set to ON for the connection.
	When `ANSI_NULLS` is ON, SQL Server enforces ANSI rules for handling `NULL` comparisons. The ANSI syntax `IS NULL` or `IS NOT NULL` must be used for all `NULL` comparisons. For example:
	`SELECT *`
	`FROM MyTable`
	`WHERE MyColumn IS NULL`
	The Transact-SQL syntax `= NULL` and `<> NULL` are not supported.
	When `ANSI_NULLS` is OFF, the Equals (=) and Not Equal To (<>) comparison operators must be used to make comparisons with `NULL` and nonnull values in a table.
	When `ANSI_WARNINGS` is ON, SQL Server generates warning messages for conditions that violate ANSI rules but do not violate the rules of Transact-SQL. For example, SQL Server will generate error and warning messages for divide-by-zero errors, string too large for database column errors and when NULL values are encountered when using aggregate functions. When SET `ANSI_WARNINGS` is OFF, these errors and warnings are not raised.
	When `ANSI_PADDING` is ON, trailing blanks on varchar values and trailing zeroes on varbinary values are not automatically trimmed.
	For more information about the `ANSI_NULLS`, `ANSI_WARNINGS`, and `ANSI_PADDING` options, see the SQL Server Transact-SQL documentation.
	By default, `AnsiNPW` is ON.

Attribute	Description

Language = *value*	The national language to use for SQL Server system messages. Use this format:
	`Language = language`
	where `language` is one the language aliases contained in the `sys.syslanguages` table.
	For example:
	`Language = French`
	If no language is specified, the connection uses the default language specified for the login on the server.

Appname = *value*	The name SQL Server uses to identify the application that connects using this data source. For example, the following entry identifies an application as `isql`: `Appname = isql` The default value is `ODBC`. SQL Server stores the application name in the `master.dbo.sysprocesses` column `program_name`. The name is returned by the `APP_NAME` function.

Attribute	Description
MARS_Connection = Yes | No	When ON (set to `Yes`), multiple active result sets (MARS) are enabled on the connection if the server is SQL Server 2005 or later. MARS allows applications to have more than one pending request per connection, and in particular, to have more than one active default result set per connection. Applications can execute other statements (for example, INSERT, UPDATE, DELETE, and stored procedure calls) while result sets are open. For example, an application might retrieve unprocessed items from an Orders table and then, while looping through the active result set, use an UPDATE statement to mark each order as processed. For non-MARS connections (`MARS_Connection` turned OFF) and earlier versions of SQL Server, applications cannot maintain multiple active statements on a connection. Applications that attempt to do this fail with the error "`connection is busy with results of another hstmt`". The application has to process or cancel all result sets from one batch before it can execute any other batch on that connection. Note that server-side cursors can be used to work around this limitation. There is a performance penalty associated with server-side cursors however. For more information about MARS, see the Microsoft article **Multiple Active Result Sets (MARS) in SQL Server 2005**. By default, `MARS_Connection` is OFF.
Logging = *value*	Whether Easysoft ODBC-SQL Server Driver logging is enabled. To enable Easysoft ODBC-SQL Server Driver logging, add a `Logging=Yes` entry to the relevant DSN section of the `odbc.ini` file.
LogFile = *value*	Use the `LogFile` attribute to specify the Easysoft ODBC-SQL Server Driver log file name and location. Ensure that the user who is running the application to be traced has write permission to the log file (and to the directory containing it).

Attribute	Description
PreserveCursor = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver preserves cursors when `SQLEndTran` commits or rolls back a transaction. By default, `PreserveCursor` is OFF, which means that cursors are closed when a transaction is committed or rolled back by using `SQLEndTran`. This behaviour can also be configured by setting `SQL_COPT_SS_PRESERVE_CURSORS` with `SQLSetConnectAttr`. For more information and a code sample, see **"SQL_COPT_SS_PRESERVE_CURSORS" on page 135**.
Wsid = *value*	The workstation ID. The default value is the host name of the machine where the ODBC application is running. SQL Server stores the workstation ID in the `master.dbo.sysprocesses` column `hostname`. The ID is returned by `sp_who` and the `HOST_NAME` function.
Version7 = Yes | No	Set to `Version7` to `Yes` if you are connecting to a SQL Server 7.0 database. When initiating the connection, the Easysoft ODBC-SQL Server Driver tries to discover the version of the SQL Server instance. Setting `Version7` to `Yes` reduces the number of steps in the discovery process for SQL Server 7.0 databases. This results in a slightly quicker connection time. By default, `Version7` is OFF (set to `No`).
ForceShiloh = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver assumes that it is connecting to a SQL Server 2000 instance and only uses the SQL Server 2000 version of TDS to communicate with the instance. By default, `ForceShiloh` is OFF (set to `No`).

Attribute	Description
ClientLB = Yes | No	Whether the Easysoft ODBC-SQL Server Driver tries to balance the load between the servers specified by the `Server` setting. The `ClientLB` setting only has an effect if you specify a primary server and additional fallback servers with `Server`. When `ClientLB` is ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver randomly selects a server to connect to. If the server is unavailable, the Easysoft ODBC-SQL Server Driver then moves sequentially through the list of other servers. When `ClientLB` OFF (set to `No`, the default), the Easysoft ODBC-SQL Server Driver tries to connect to the servers in the order that they are defined in. (Primary server first and then each additional fallback server.) **Example** You specify a primary server (`sqlsrvhostA`) and two fallback servers (`sqlsrvhostB` and `sqlsrvhostC`): `Server = sqlsrvhostA,sqlsrvhostB,sqlsrvhostC:1583` When `ClientLB` is ON, the Easysoft ODBC-SQL Server Driver will randomly choose a server to connect to. If, for example, the driver tries to connect to `sqlsrvhostB` first, it will then try to connect to `sqlsrvhostC` (if `sqlsrvhostB` is unavailable) and `sqlsrvhostA` (if `sqlsrvhostC` is unavailable). When `ClientLB` is OFF, the Easysoft driver will try to connect to `sqlsrvhostA` and then `sqlsrvhostB` (if `sqlsrvhostA` is unavailable) and finally `sqlsrvhostC` (if `sqlsrvhostB` is unavailable).
Failover_Partner = *value*	Use `Failover_Partner` to specify the current mirror database server. If the **initial** connection to the principal database server fails, the Easysoft ODBC-SQL Server Driver will attempt a connection to the server specified by `Failover_Partner`. For more information about database mirroring, see **"Database Mirroring" on page 263**.

Attribute	Description
MultiSubnetFailover = Yes | No	Set `MultiSubnetFailover` to `Yes` when connecting to a SQL Server Failover Cluster Instance or the availability group listener of a SQL Server availability group. Setting `MultiSubnetFailover` to `Yes` provides faster detection of and connection to the (currently) active server. By default, `MultiSubnetFailover` is OFF (set to `No`).
ApplicationIntent = ReadOnly | ReadWrite	Specifies the application workload type when connecting to a SQL Server Failover Cluster Instance or the availability group listener of a SQL Server availability group. The default is `ReadWrite`.
ConnectRetryCount = *num*	Specifies the number of reconnection attempts if there is a connection failure. Valid values range from 0 to 255. Zero (0) means do not attempt to reconnect. The default value is one reconnection attempt.
ConnectRetryInterval = *num*	Specifies the number of seconds between each connection retry attempt. Valid values are 1-60. The default value is 10 seconds,
VarMaxAsLong = Yes | No	When OFF (set to `No`), the Easysoft ODBC-SQL Server Driver returns a `varchar(max)` column as a `SQL_VARCHAR` with a zero length, which means the maximum size is unlimited. Some applications may interpret this to mean that the column size is zero bytes rather than unlimited and allocate a buffer that is too small for the column data. To work around this, try setting `VarMaxAsLong` to `Yes`. When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver returns a `varchar(max)` column as a `SQL_LONGVARCHAR`. By default, `VarMaxAsLong` is OFF.

Attribute	Description
VarMaxAsVarchar = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver maps a `varchar(max)` column to a `varchar(4000)` column and an `nvarchar(max)` column to a `nvarchar(4000)` column. By default, `VarMaxAsVarchar` is OFF.
DisguiseGuid = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver describes the `UNIQUEIDENTIFIER` data types as `CHAR` rather than `GUID`. This is a workaround for applications such as Oracle's HSODBC that do not recognise `UNIQUEIDENTIFIER` types and therefore fail to return data from tables containing these column types. By default, `DisguiseGuid` is OFF (set to `No`).
DisguiseLong = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver describes `IMAGE` and `TEXT` data types as `VARBINARY` and `VARCHAR`. This is a workaround for applications such as Oracle's HSODBC that cannot handle `IMAGE` and `TEXT` types and therefore fail to return data from tables containing these column types. By default, `DisguiseLong` is OFF (set to `No`).
LimitLong = *num*	The maximum size in bytes that the Easysoft ODBC-SQL Server Driver returns for `image`, `ntext`, `text`, `nvarchar(max)`, `varbinary(max)` and `varchar(max)` columns. Use `LimitLong` to restrict the size returned by the driver when describing these data types. **Note** `LimitLong` only has an effect on `MAX` data types if `VarMaxAsLong` is ON (set to `Yes`). By default, `LimitLong` is OFF (set to `No`).

Attribute	Description
DPrec = *num*	The precision to use when converting `float(25-53)` data in a result set to a string If an application specifies a string as the target type for non-character data in a `SQLBindCol` or `SQLGetData` call, the Easysoft ODBC-SQL Server Driver converts the data to the target type. Use the `FPrec` attribute to specify the precision to use when the driver does this conversion for `float(25-53)` data. The default precision is 6.
FPrec = *num*	The precision to use when converting `float(1-24)` or `real` data in a result set to a string If your application specifies a string as the target type for non-character data in a `SQLBindCol` or `SQLGetData` call, the Easysoft ODBC-SQL Server Driver converts the data to the target type. Use the `FPrec` attribute to specify the precision to use when the driver does this conversion for `float(1-24)` or `real` data. The default precision is 6.

Attribute	Description

Strftime = *format*	The format to use when converting ftimestamp data in a result set to a string.
	If your application specifies a string as the target type for timestamp data in a `SQLBindCol` or `SQLGetData` call, the Easysoft ODBC-SQL Server Driver converts the data to the target type. Use the `Strftime` attribute to specify the format to use when the driver does this conversion for `date`, `datetime`, `datetime2`, `datetimeoffset`, `smalldatetime`, and `time` data.
	The Easysoft ODBC-SQL Server Driver uses `strftime` to do the conversion, and so *format* should be one of the format strings supported by `strftime`. For the available format strings, see the `strftime(3)` man page.
	For example, the format string specified in the following line:
	`STRFTIME = %d %h %Y %T`
	would produce:
	`11 May 2011 12:35:29`
	given this SQL statement:
	`SELECT CAST('2011-05-11 12:35:29.123' AS datetime)`

Strfsize = *num*	The display size of a column is the maximum number of characters needed to display data in character form. It may be necessary to increase the default display size for timestamp data to accommodate some of the formats that `strftime` supports. For example, to set the display size to 32, add the following line to your data source: `STRFSIZE = 32`

Attribute	Description

ConvToUtf = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver converts UCS-2 encoded data to UTF-8 and vice versa. This enables applications running on UTF-8 platforms to work with Unicode data stored in `nchar`, `nvarchar`, `nvarchar(max)` and `ntext` columns.
	SQL Server uses UCS-2 to encode data in `nchar`, `nvarchar`, `nvarchar(max)` and `ntext` columns. If your application expects UTF-8 encoded data, and is unable to convert data to this encoding scheme, it will be unable to process Unicode data stored in `nchar`, `nvarchar`, `nvarchar(max)` and `ntext` columns. To work around this, add this line to your ODBC data source.
	`ConvToUtf = 1`
	`ConvToUtf` also affects SQL statement text, metadata (table names and so on), and SQL statement parameters that are bound as a wide type (`SQL_WCHAR`, `SQL_WVARCHAR`, `SQL_WLONGVARCHAR`). For example:
	`SQLPrepare( hstmt, "INSERT INTO MYNCHARTABLE VALUES (?)", SQL_NTS );`
	`SQLBindParameter( hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_WCHAR, 100, 0, &cval, sizeof( cval ), &len1 );`
	By default, `ConvToUtf` is OFF (set to `No`).
	Example: Retrieving Data
	We ran OpenOffice.org 2.0 on Ubuntu from a shell in which the `LANG` environment was set to `en_GB.UTF-8`. With `ConvToUtf` set to `No`, we connected to a SQL Server data source in OOo Base and ran this SQL:

Attribute	Description

	`use Northwind`
	`select CompanyName from Suppliers where SupplierID = 29`
	SQL Server stores data in the `CompanyName` column as a UCS-2 encoded `nvarchar` type.
	The results for this query should be:
	`Forêts d'érables`
	Instead, we got these results:
	`For?ts d'?rables`
	The `?` symbols indicate that application was unable to convert the character from the server encoding scheme to the client encoding scheme.
	In OOo Writer, we used the **Insert > Special Character** command to insert `ê` and `é` into a new document. We did this to show that OpenOffice.org running on this system and environment was capable of rendering these two characters. We then saved the document as a Text file and ran the following command at the shell prompt:
	`$ file ooo_chars.txt`
	`ooo_chars.txt: Unicode text, UTF-8`
	The file command's output indicates that the encoding scheme OpenOffice was using is UTF-8.
	We set `ConfToUTF` to `Yes` and reconnected to the data source in Base. Running the same query returned the expected results. This is because the Easysoft ODBC-SQL Server Driver converts the UCS-2 encoded data to UTF-8, the encoding OpenOffice expects.

Attribute	Description

	Example: Inserting Data
	We created a SQL file named `insert-northwind-shipper.sql` on a Ubuntu machine:
	`-- Insert new record into the Northwind shippers table`
	`USE Northwind;`
	`INSERT INTO Shippers (CompanyName, Phone) VALUES (N'♦ Diamond Shipping', '(11) 555-2167');`
	`SELECT * FROM Shippers;`
	To create the file, we used the Vi IMproved (vim) text editor from a shell in which the `LANG` environment was set to `en_GB.UTF-8`. To insert the ♦ character in vim, we typed CTRL+V u2666. (u+2666 is the Unicode code point for this character.) The `N` prefix before the INSERT statement value tells SQL Server that the string contains a Unicode character. To confirm that the SQL file was UTF-8 encoded, we ran the `file` command:
	`$ file insert-northwind-shipper.sql`
	`insert-northwind-shipper.sql: Unicode text, UTF-8`
	In the same shell, we used `insert-northwind-shipper.sql` as an input file to `isql`:
	`/usr/local/easysoft/unixODBC/bin/isql -v SQLSERVER_SAMPLE < insert-northwind-shipper.sql`
	The `SQLSERVER_SAMPLE` data source connects to a SQL Server instance that serves the Northwind database. In the data source, `ConfToUTF` was set to `Yes`. The command's output confirmed that the new record had been successfully inserted and that the Ubuntu machine was capable of rendering the ♦ character:
	`SQL>--+---------------------+----------------------+`
	`| ID | CompanyName | Phone |`
	`+-----+----------------------+-------------------------+`
	`| 1 | Speedy Express | (503) 555-9831 |`
	`| 2 | United Package | (503) 555-3199 |`
	`| 3 | ♦ Diamond Shipping | (11) 555-2167 |`
	`+------+----------------------+-------------------------+`

Attribute	Description
ConvWToUtf = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver converts strings passed to Unicode ODBC calls (with suffix "W") to UTF-8. The Easysoft ODBC-SQL Server Driver also converts metadata and result sets returned by Unicode ODBC calls to UTF-8. By default, `ConvWToUtf` is OFF (set to `No`).

SQLServerUTF = Yes | No	When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver sets the `ConvWToUtf` attribute to `Yes` prior to connecting to the data source. This provides a workaround for applications that pass UTF-8 encoded strings to `SQLConnectW`, `SQLDriverConnectW` and `SQLBrowseConnectW`.
	The `SQLServerUTF` attribute must be specified in a section named `ODBC` in `odbc.ini`. For example:
	`[ODBC]`
	`SQLServerUTF = Yes`
	By default, `SQLServerUTF` is OFF (set to `No`).

UTF8DB = Yes | No	If you have set a UTF-8 collation at instance, database or column level, add this entry to your data source: `UTF8DB = Yes` UTF-8 collations were introduced in SQL Server 2019 and have the suffix `UTF8`.

Attribute	Description

Client_CSet = *encoding*	Specifies the encoding on the Easysoft ODBC-SQL Server Driver machine.
	If set, the Easysoft ODBC-SQL Server Driver tries to convert to and from the specified encoding when retrieving and submitting character data. For example, if you have an application that is expecting `EUC-JP` encoded character data, you would need to set `Client_CSet` to specify the `EUC-JP` encoding:
	`# Convert from EUC-JP when submitting data to SQL Server`
	`# Convert to EUC-JP when retrieving data from SQL Server`
	`Client_CSet = EUC-JP`
	All character data is affected by `Client_CSet`, including data stored in `char`, `varchar`, `text`, `nchar`, `nvarchar`, `nvarchar(max)` and `ntext` columns, metadata (table names and so on) and SQL statement text and parameters.
	Use `Client_CSet` if you experience data loss/corruption when working with character data *and* your application cannot convert data to the encoding scheme it expects.
	The Easysoft ODBC-SQL Server Driver uses a built-in version of iconv to do the conversion. For a list of available encodings for `Client_CSet`, run this command on the machine where the Easysoft ODBC-SQL Server Driver is installed:
	`iconv -l`
	Set `Client_CSet` to the encoding that corresponds with the `LANG` environment variable value on the client machine. For example, if `LANG` was set to `en_US.UTF-8` on the client machine, you would set `Client_CSet` to `UTF-8`.
	If iconv cannot convert a character, the Easysoft ODBC-SQL Server Driver will omit the character and write this entry to the unixODBC or driver log file (assuming logging is enabled):
	`One or more characters in the input stream could not be converted`

Attribute	Description
	Note that if your client machine encoding is UTF-8, you can use `Client_CSet` as an alternative to `ConvToUTF`. You do not need to set both data source attributes. If you specify a `Server_CSet` value without specifying a `Client_CSet` value, the Easysoft ODBC-SQL Server Driver uses `ISO8859-1` as the client machine encoding.

Server_CSet = *encoding*	Specifies the SQL Server encoding for non-Unicode character data.
	If set, the Easysoft ODBC-SQL Server Driver tries to convert character data *from* the specified encoding when retrieving SQL Server data stored in `char`, `varchar` and `text` columns. The Easysoft ODBC-SQL Server Driver also tries to convert strings *to* the specified encoding when binding parameter markers. For example:
	`INSERT INTO my_table(my_char_col, my_varchar_col, my_text_col) VALUES (?,?,?)`
	`SELECT * FROM my_table WHERE my_varchar_col = ?`
	Use `Server_CSet` if you experience data loss/corruption when working with data stored in `char`, `varchar` and `text` columns.
	The Easysoft ODBC-SQL Server Driver links to iconv on your machine at run-time to do the conversion.
	Set `Server_CSet` to the iconv encoding that corresponds with the SQL Server code page. To find out the SQL Server code page, run:
	`SELECT COLLATIONPROPERTY('collation' , 'CodePage') AS CodePage`
	where *collation* is the SQL Server database collation, if set, otherwise the SQL Server instance collation.

Attribute	Description

	In the following example, *collation* would be `Cyrillic_General_CI_AS`.
	`SELECT DATABASEPROPERTYEX('MyDatabase', 'Collation') SQLCollation;`
	`SQLCollation`
	`------------`
	`NULL`
	`SELECT SERVERPROPERTY('Collation') SQLCollation;`
	`SQLCollation`
	`------------`
	`Cyrillic_General_CI_AS`
	For a list of iconv encodings, run this command on the machine where the Easysoft ODBC-SQL Server Driver is installed:
	`iconv -l`
	As an example, if the SQL Server collation was `Cyrillic_General_CI_AS`, the associated code page is 1251, and you would set `Server_CSet` to `WINDOWS-1251`.
	If you set the `Client_CSet` attribute without setting the `Server_CSet` attribute, the Easysoft ODBC-SQL Server Driver uses the `ISO8859-1` encoding as the `Server_CSet` value.

Server_UCSet = *encoding*	Specifies the SQL Server encoding for character data. If set, the Easysoft ODBC-SQL Server Driver tries to convert character data *from* the specified encoding when retrieving SQL Server data stored in `nchar`, `nvarchar` and `ntext` columns. The Easysoft ODBC-SQL Server Driver also tries to convert strings *to* the specified encoding when binding parameter markers. The default Unicode encoding is `UTF-16le`.

Attribute	Description
Use_LCID = Yes | No	Whether to automatically work out which character set to use based on the SQL Server column or instance Locale ID (LCID). If set, the Easysoft ODBC-SQL Server Driver tries to convert character data *from* the character set when retrieving SQL Server data stored in `char`, `varchar` and `text` columns. The Easysoft ODBC-SQL Server Driver also tries to convert strings *to* the character set when inserting data into `char`, `varchar` and `text` columns. The ODBC application must bind the character data as a Unicode data type (for example, a `SQL_WCHAR`). This feature emulates the Microsoft Native Client ODBC Driver's automatic translation of character data, and, as is the case with the Microsoft driver, not all character sets are supported. If your character set is not supported, the `Client_CSet` and `Server_CSet` attributes provide an alternative conversion mechanism. By default `Use_LCID` is OFF (set to `No`).
LCID = *localeid*	Sets the `ClientLCID` the parameter in the TDS login packet. The `ClientLCID` parameter is described in the TDS protocol specification: **http://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-TDS%5D.pdf**

Attribute	Description

Trusted_Domain = *value*	The Windows domain that the user specified with `User` belongs to.
	If the user belongs to the same domain as the one that the SQL Server machine is in, you can omit `Trusted_Domain`. The Easysoft ODBC-SQL Server Driver automatically detects the domain in this case.
	If you specify a Domain with `Trusted_Domain`, set `Trusted_Connection` to `Yes` and omit the domain from `User`. For example:
	`# Windows authentication`
	`User = mylocalmachineuser`
	`Password = mypassword`
	`Trusted_Connection = 1`
	`Trusted_Domain = mymachinename`

Trusted_Connection = Yes | No	Whether to use Windows or SQL Server authentication to validate the connection.
NTLMv2 = Yes | No	If you want to use NTLMv2 to authenticate the Windows user specified with `User`, set set `NTLMv2` to Yes. Otherwise, leave `NTLMv2` set to its default value `No` (OFF).
IPv6 = Yes | No	Set `IPv6` to `Yes` when connecting to a SQL Server 2005 or later instance that is listening on an IPv6 address. By default, IPv6 is OFF (set to `No`), which means that the Easysoft ODBC-SQL Server Driver assumes that the target SQL Server instance is listening on an IPv4 address. For more information about IPv6, see **"Connecting to SQL Server 2005 or Later by Using IPv6" on page 274**.

Attribute	Description
ConnectionTimeout = *num*	The number of milliseconds to wait for any request on the connection to complete before returning to the application. After the initial connection to the SQL Server machine has been established, the Easysoft ODBC-SQL Server Driver will wait *num* milliseconds each time it needs a response from SQL Server. If no response is received from SQL Server before the timeout expires, the Easysoft ODBC-SQL Server Driver returns the error `Timeout expired`. The default value 0 means that no connection timeout is applied by the Easysoft ODBC-SQL Server Driver. A timeout set by calling `SQLSetConnectAttr` with the `SQL_ATTR_CONNECTION_TIMEOUT` connection attribute will override `ConnectionTimeout`.

Attribute	Description
LogonTimeout = *num*	The number of milliseconds to wait for a TCP connection to the SQL Server machine to be established before returning to the application. When you define a timeout, the initial connection phase lasts for *num* milliseconds. If the Easysoft ODBC-SQL Server Driver is unable to connect to the target SQL Server machine before the timeout expires, it returns the message `Connection timeout expired`. Note that if you specify a named instance in the `Server` attribute value, the driver returns a different timeout message: `Failed to get datagram from socket`. The default value 0 means that no initial connection timeout is applied by the Easysoft ODBC-SQL Server Driver. The Easysoft ODBC-SQL Server Driver classes the connection phase as obtaining the IP address of the SQL Server machine and connecting to it. This means that if you specify the `Server` attribute value as a machine name rather than an IP address, your system resolver library will be used (possibly examining `/etc/hosts` or doing a DNS query). On some operating systems, `gethostbyname()`, the call used to resolve a machine name into an IP address, cannot be interrupted and the connection timeout will not work. If this is a problem for you, either specify the SQL Server machine as an IP address or tell your resolver library to consult `/etc/hosts` before DNS and place an entry in `/etc/hosts`. A timeout set by calling `SQLSetConnectAttr` with the `SQL_ATTR_LOGIN_TIMEOUT` connection attribute will override `LogonTimeout`.

Attribute	Description
RcvBuffer = *num*	The size of the receive buffer for the socket in bytes. Possible values for *num* are: 0, do not set the receive buffer size, use the system default value. *n*, where *n* is a number greater than 0, set the receive buffer to the specified size by passing *n to* the `setsockopt()` function . By default, the system default receive buffer size is used.
SoKeepalive = Yes | No	Whether to use TCP keepalive probes to verify that an idle connection is still intact. When ON (set to `Yes`), keepalive probes are sent, after a period of inactivity, to verify that the connection to the SQL Server machine is still valid. To do this, the Easysoft ODBC-SQL Server Driver sets the `SO_KEEPALIVE` socket option by using `setsockopt()`. If no response to the probes is received, the socket is closed. The duration of the period of inactivity is a system default, and is typically two hours. By default, `SoKeepalive` is OFF (set to `No`).
PacketSize = *num*	The TDS packet size in bytes that the Easysoft ODBC-SQL Server Driver will request. The specified packet size must be lower than 65536 bytes. The default packet size is 4096 bytes.
ColumnEncryption = Enabled | Disabled	Set this attribute to `Enabled` if you want to query or update data held in an Always Encrypted column. You will also need to set the `Driver` attribute to `Easysoft ODBC-SQL Server SSL`. Always Encrypted columns were introduced in SQL Server 2016. For more information, see: **http://www.easysoft.com/blog/sql-server-always-encrypted.html**

Attribute	Description

Allow_C_Comment = Yes | No	SQL comments are nonexecuting text strings used to document or temporarily disable SQL statements. SQL Server supports comments preceded by double hyphens (`--`) or delimited by forward slash-asterisk character pairs (`/* ... */`).
	By default, the Easysoft ODBC-SQL Server Driver strips single line `/* ... */` comments from SQL statements before passing the SQL to SQL Server. For example:
	`SELECT ContactID, /* FirstName, */ LastName FROM Person.Contact`
	becomes:
	`SELECT ContactID, LastName FROM Person.Contact`
	To preserve single line `/* ... */` comments in the SQL that is passed to SQL Server, set the `Allow_C_Comment` attribute to `Yes`.

XATimeout = *num*	Sets the timeout for an XA transaction created by an `xa_open` call initiated by the Easysoft ODBC-SQL Server Driver. For more information about the driver's XA support, see the Easysoft blog on the Easysoft web site.

Attribute	Description
Kerberos = Yes | No	Whether to access a SQL Server instance as a Kerberos service. When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver will attempt to obtain a service ticket for the following Service Principal Name (SPN): `MSSQLSvc/`*`server`*`:`*`port`* where: *server* is the name or IP address of the SQL Server machine specified with the `Server` attribute. *port* is the port on which the SQL Server instance is listening, which is specified with the `Port` attribute. The `ServerSPN` attribute provides an alternative way to supply a service principal name. Do not specify a `User` or `Password` value in the data source if you set `Kerberos` to `Yes`. The Kerberos application `kinit` must have already been used for authentication on the Easysoft ODBC-SQL Server Driver machine. For more information about `kinit` and accessing SQL Server as a Kerberos service, see the following Easysoft tutorial: **http://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html** By default, `Kerberos` is OFF (set to `No`), and the Easysoft ODBC-SQL Server Driver will use either SQL Server or Windows authentication (see **"SQL Server Authentication Modes" on page 237**) to validate a user specified in the data source.

Attribute	Description
ServerSPN = *value*	The Service Principal Name (SPN) for a SQL Server instance that has been registered as a Kerberos service. Your database administrator will have registered an SPN for your SQL Server instance. Contact your database administrator for the SPN value and then specify that value with `ServerSPN`. As an alternative to the `ServerSPN` attribute, you can set the `Kerberos` attribute to 1 and the Easysoft ODBC-SQL Server Driver will build an Service Principal Name from the `Server` and `Port` attribute values. If the SPN contains an instance name (for example, `MSSQLSvc/mysqlservermachine:myinstance`), you need to use the `ServerSPN` attribute rather than the `Kerberos` attribute. Do not specify a `User` or `Password` value in the data source if you specify a `ServerSPN` value. **Windows** This attribute is labelled SPN.
FailoverServerSPN = *value*	The SPN for a mirror server instance that has been registered as a Kerberos service. For more information about Database Mirroring see **"Database Mirroring" on page 263**. For more information about Kerberos and Database Mirroring, see the Easysoft tutorial: **http://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html**

Attribute	Description

GSSLib = *value*	The Easysoft ODBC-SQL Server Driver uses `libgssapi_krb5.so`, the Kerberos GSS-API library, to request service tickets for accessing SQL Server instances. If the Easysoft ODBC-SQL Server Driver is unable to open this library, the connection will fail with the error:
	`Krb5: failed to open gss lib (libgssapi_krb5.so)`
	If the Kerberos GSS-API library is not called `libgssapi_krb5.so` in your GSS-API distribution, use the `GSSLIB` attribute in your data source to specify the alternative GSS-API library. For example:
	`GSSLIB = /opt/extension/lib/libgssapi.so`

GSSHost = Yes | No	Whether the Easysoft ODBC-SQL Server Driver allows the use of `GSS_C_NT_HOSTBASED_SERVICE` or `GSS_C_NT_USER_NAME` as the target principal name. When ON (set to `Yes`), the Easysoft ODBC-SQL Server Driver allows the use of `GSS_C_NT_HOSTBASED_SERVICE`. By default, `GSSHOST` is OFF (set to `No`).

Attribute	Description

GSSFlag = *req_flags*	The Easysoft ODBC-SQL Server Driver allows you to pass `req_flags` to the `gss_init_sec_context()` function, which is used to initiate a security context for the driver. The Key Distribution Center (KDC) uses this security context to verify the identity of the client. To pass `req_flags` to `gss_init_sec_context()`, use the `GSSFLAG` attribute:
	`GSSFLAG = req_flags`
	where `req_flags` is a bitmask specifying the requested GSS services. To look up the available bitmask values, refer to the `gssapi.h` header file for the GSS-API distribution on the Easysoft ODBC-SQL Server Driver machine. The driver default `GSSFLAG` value is 4, which sets the `GSS_C_REPLAY_FLAG` flag.
	As an example, to request credential delegation, set the `GSS_C_DELEG_FLAG` flag by including this line in your data source:
	`GSSFLAG = 1`

Figure 11: Easysoft ODBC-SQL Server Driver data source settings.

ODBC Driver Manager Attribute Fields

The following attributes may be set in the `ODBC` section of the `odbc.ini` file:

Attribute	Description

SQLServerUTF = Yes | No	When ON (set to `Yes`), the `ConvWToUtf` attribute is set to `Yes` prior to connecting to the data source. This provides a workaround for applications that pass UTF-8 encoded strings to `SQLConnectW`, `SQLDriverConnectW` and `SQLBrowseConnectW`.
	The `SQLServerUTF` attribute must be specified in a section named `ODBC` in `odbc.ini`. For example:
	`[ODBC]`
	`SQLServerUTF = Yes`
	By default, `SQLServerUTF` is OFF (set to `No`).

Figure 12: ODBC Driver Manager attribute fields.

ENVIRONMENT

The Easysoft ODBC-SQL Server Driver must be able to find the following shared objects, which are installed during the Easysoft ODBC-SQL Server Driver installation:

- `libodbcinst.so`

  By default, this is located in `/usr/local/easysoft/unixODBC/lib`.

- `libeslicshr.so`

  By default, this is located in `/usr/local/easysoft/lib`.

- `libessupp.so`

  By default, this is located in `/usr/local/easysoft/lib`.

- `libestdscrypt.so`

  By default, this is located in `/usr/local/easysoft/lib`.

  For more information about `libestdscrypt.so`, see **"Windows Authentication" on page 238**.

You may need to set and export `LD_LIBRARY_PATH`, `SHLIB_PATH` or `LIBPATH` (depending on your operating system and run-time linker) to include the directories where `libodbcinst.so`, `libeslicshr.so` and `libessupp.so` are located.

Note The shared object file extension (`.so`) may vary depending on the operating system (`.so`, `.a` or `.sl`).

ESTABLISHING A TEST CONNECTION

The `isql` query tool lets you test your Easysoft ODBC-SQL Server Driver data sources.

To test the Easysoft ODBC-SQL Server Driver connection

1. Change directory into `/usr/local/easysoft/unixODBC/bin`.
2. Type `./isql -v` *`data_source`*, where *data_source* is the name of the target data source.

   If you are unable to connect, see **"Troubleshooting Database Connection Problems" on page 122** for help on solving some common connection problems.
3. At the prompt, type an SQL query. For example:

   ```
   SQL> select * from mytable;
   ```

   – OR –

   Type `help` to return a list of tables:

   ```
   SQL> help
   ```

TROUBLESHOOTING DATABASE CONNECTION PROBLEMS

This section lists some common connection problems and their solutions.

- **Client unable to establish connection: OS Error: 'Failed to find host address 'myhost\myinstance'**
- **Client unable to establish connection: OS Error: 'Connection refused'**
- **Client unable to establish connection: Server not configured for TCP connection**
- **Client unable to establish connection: OS Error: 'Failed to get datagram from socket'**
- **Login failed for user ''. The user is not associated with a trusted connection**
- **Login failed for user 'myuser'.**

Client unable to establish connection: OS Error: 'Failed to find host address 'myhost\myinstance'

Check the Server attribute in your data source specifies a valid machine name or IP address. Check that the machine name can be looked up by using DNS or is present in `/etc/hosts`. Check that you are on the same network as the target host by pinging the machine:

```
ping myhost
```

If `ping` times out or fails, then either the DNS lookup is not working properly or there is some other networking or routing issue that needs to be resolved. Contact your network administrator.

Client unable to establish connection: OS Error: 'Connection refused'

Check that the SQL Server instance that you are trying to connect to is running.

On the SQL Server machine, "SQL Server <*instance*>" will be listed in output of the `net start` command, if the SQL Server instance is running.

If SQL Server is listening on a fixed TCP port, check that you can use telnet to connect to the port that you have specified in the data source:

```
telnet hostname port
```

where *hostname* is the host name or IP address of the machine where SQL Server is running and *port* is the port number that you have specified with the `Port` attribute. If the SQL Server instance is listening on this port, you will see output similar to:

```
Connected to myserver

Escape character is '^]'
```

To exit from telnet, type CTRL-] and then `quit`.

If you do not see this output or a "Connection refused" error displays, SQL Server is not listening on the specified port. Contact your database administrator for the correct SQL Server port.

If you are using the correct port but are unable to connect with telnet, the SQL Server instance may not allow remote TCP/IP connections. See **"Client unable to establish connection: Server not configured for TCP connection" on page 124**.

Client unable to establish connection: Server not configured for TCP connection

The TCP/IP protocol must be enabled in the instance that you are trying to connect to.

In the SQL Server Configuration Manager, in the list of network protocols for the instance, the status for TCP/IP must be set to "Enabled".

By default, SQL Server 2005 and later do not allow remote connections, which means that the default setting for TCP/IP is "Disabled".

Client unable to establish connection: OS Error: 'Failed to get datagram from socket'

The Easysoft ODBC-SQL Server Driver uses the SQL Server Browser or the SQL Server 2000 listener service to find out what TCP port SQL Server is listening on. If the SQL Server Browser or listener service is not running and active, the Easysoft ODBC-SQL Server Driver will be unable to open a connection for this purpose and the "`Failed to get datagram from socket`" error displays.

On the SQL Server machine, "SQL Server Browser" will be listed in output of the `net start` command, if the SQL Server Browser is running. If `net start` shows that the SQL Server Browser service is running, the service may not be active. In the SQL Server Configuration Manager, the Active option must be set to "Yes" in the Advanced SQL Server Browser property tab. (The SQL Server Browser service must be restarted before any change to this setting takes effect.)

If you are connecting to SQL Server through a firewall, the firewall needs to allow connections through:

- The SQL Browser UDP port, 1434.
- The TCP port that the SQL Server instance is listening on.

If UDP port 1434 is not open, the firewall will block the connection when the Easysoft ODBC-SQL Server Driver attempts to discover the SQL Server port and the 'Failed to get datagram from socket' will display.

Because the SQL Server Browser or listener accepts unauthenticated UDP requests, it may have been turned off as a security measure, and your database administrator will have configured each SQL Server instance to listen on a specific TCP port. You need to specify this port number with the `Port` setting. For example, if SQL Server is listening on port 1500, add this line to the data source in `odbc.ini`:

```
Port = 1500
```

The "`Failed to get datagram from socket`" error also displays if you try to connect to a hidden SQL Server instance. You need to specify the port that the hidden instance is listening even though the SQL Server Browser or listener may be running.

Login failed for user ''. The user is not associated with a trusted connection

Check that the `User` and `Password` attributes for the data source in the `odbc.ini` specify a valid Windows user name and password.

This error also displays if you try to connect to SQL Server with a SQL Server user name and password but SQL Server's authentication mode is set to Windows Authentication only. To connect by using a SQL Server account, the security mode for the SQL Server instance must be changed to mixed (both SQL Server and Windows authentication are enabled).

To enable mixed mode, your database administrator must set the SQL Server security property **Server Authentication** to **SQL Server and Windows Authentication mode**. Note that Microsoft recommend that Windows authentication is used to connect to SQL Server whenever possible.

Login failed for user 'myuser'.

Check that the `User` and `Password` attributes for the data source in the `odbc.ini` specify a valid SQL Server user name and password.

This error also displays if you try to connect to SQL Server with a valid Windows user name and password but no corresponding SQL Server login exists. For example, SQL Server Setup creates a login named BUILTIN\Administrators that allows members of the local Administrators Windows group to access SQL Server. As a security measure, the database administrator may delete this login and members of this group will then need individual SQL Server login accounts to access SQL Server.

Ask your database administrator to create a SQL Server login for you that uses Windows authentication to validate your connection details.

DSN-less Connections

In addition to using a data source, you can also connect to a database by using a DSN-less connection string of the form:

```
SQLDriverConnect(..."DRIVER={Easysoft ODBC-SQL
Server};
                Server=server;UID=user;PWD=password;
                Port=port;"...)
```

where *server* is the SQL Server instance that you want to connect to, *user* and *password* are a valid SQL Server login and password and *port* is the TCP port that SQL Server is listening on. You need to use the `Easysoft ODBC-SQL Server` DRIVER keyword to identify the Easysoft ODBC-SQL Server Driver.

Other Easysoft ODBC-SQL Server Driver attribute settings, as described in **"Setting Up Data Sources on Unix" on page 69**, can be added to the connection string using the same `PARAMETER=value;` format. For example, the following connection string changes the default database with the `Database` attribute.

Linux/UNIX example:

```
"DRIVER={Easysoft ODBC-SQL
Server};Server=myhost\\SQLEXPRESS;UID=mydomain\\myuser;PWD=mypassword;Port=1500
;Database=Sales;"
```

Windows example:

```
"DRIVER={Easysoft ODBC-SQL Server
Driver};Server=myhost\SQLEXPRESS;UID=sa;PWD=mypassword;Database=Sales;"
```

APPENDIX A TECHNICAL REFERENCE

Technical Reference for the Easysoft ODBC-SQL Server Driver

This section contains extra information relating to the deployment of the Easysoft ODBC-SQL Server Driver.

Appendix Guide

ODBC Conformance

The Easysoft ODBC-SQL Server Driver complies with the ODBC 3.81 specification.

The Easysoft ODBC-SQL Server Driver is Level 2 compliant.

ODBC API Support

All ODBC 3.81 calls are supported.

Cursor Support

The Easysoft ODBC-SQL Server Driver supports FORWARD_ONLY, KEYSET_DRIVEN, DYNAMIC and STATIC cursors.

Supported Data Types

The Easysoft ODBC-SQL Server Driver supports the following SQL Server data types:

- `bigint`
- `binary`
- `bit`
- `char`
- `date`
- `datetime`
- `datetime2`

- datetimeoffset
- decimal
- float
- geography
- geometry
- hierarchyid
- image
- int
- money
- numeric
- real
- smalldatetime
- smallint
- smallmoney
- sql_variant
- sysname
- text
- time
- timestamp
- tinyint
- uniqueidentifier
- varbinary
- varbinary(max)

- `varchar`
- `varchar(max)`
- `xml`

Notes

The `varchar(max)`, `nvarchar(max)`, `varbinary(max)` and `xml` data types were introduced in SQL Server 2005.

The `date`, `datetime2`, `datetimeoffset`, `geography`, `geometry`, `hierarchyid` and `time` data types were introduced in SQL Server 2008.

The Easysoft ODBC-SQL Server Driver lets you insert, update, and delete FILESTREAM data by using SQL. SQL Server stores FILESTREAM data on the file system rather than in the database file. To specify that the data should be stored externally, the FILESTREAM column attribute must be set. The FILESTREAM attribute was introduced in SQL Server 2008, and applies to `varbinary(max)` columns.

The SQLGetTypeInfo Function

SQL Server treats identity as an attribute, whereas ODBC treats it as a data type. To resolve this mismatch, `SQLGetTypeInfo` returns the data types: `int identity`, `smallint identity`, `tinyint identity`, `decimal() identity`, and `numeric() identity`. The `SQLGetTypeInfo` result set column `AUTO_UNIQUE_VALUE` reports the value `TRUE` for these data types.

For `varchar`, `nvarchar` and `varbinary` data types, the Easysoft ODBC-SQL Server Driver continues to report 8000, 4000 and 8000 for the `COLUMN_SIZE` value, even though it is actually unlimited. This is to ensure backward compatibility.

For the `xml` data type, the Easysoft ODBC-SQL Server Driver reports `SQL_SS_LENGTH_UNLIMITED` for `COLUMN_SIZE` to denote unlimited size.

THE SQLSETCONNECTATTR FUNCTION

The Easysoft ODBC-SQL Server Driver supports a number of driver-specific ODBC connection attributes. These are defined in `/usr/local/easysoft/sqlserver/include/sqlncli.h`. The Easysoft ODBC-SQL Server Driver may require that the attribute be set prior to connection, or it may ignore the attribute if it is already set:

Attribute	Set before or after connection to server
SQL_COPT_SS_INTEGRATED_SECURITY	Before
SQL_COPT_SS_PRESERVE_CURSORS	Before
SQL_COPT_SS_TXN_ISOLATION	Either

SQL_COPT_SS_INTEGRATED_SECURITY

Whether to use Windows or SQL Server authentication to validate the connection.

Value	Description
SQL_IS_OFF	Default. Use SQL Server Authentication to authenticate the connection.
SQL_IS_ON	Use Windows Authentication to authenticate the connection.

Windows authentication examples:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>
.
.
.
 /* Use Windows Authentication to validate the connection  */
SQLSetConnectAttr(dbc, SQL_COPT_SS_INTEGRATED_SECURITY,
                  (void *) SQL_IS_ON, 0);
/* Specify a Windows user name and password. mywindowsuser belongs to the */
/*  same domain as the SQL Server machine, so there is no need to specify it */
/* - the Easysoft ODBC-SQL Server Driver will automatically detect the domain */
SQLDriverConnect(dbc, NULL, "DRIVER={Easysoft ODBC-SQL
Server};SERVER=myserver\\SQLEXPRESS;UID=mywindowsuser;PWD=mywindow
spassword",
                 SQL_NTS, outstr, sizeof(outstr), &outstrlen,
                 SQL_DRIVER_COMPLETE);
                – OR –
SQLSetConnectAttr(dbc, SQL_COPT_SS_INTEGRATED_SECURITY,
                  (void *) SQL_IS_ON, 0);
SQLDriverConnect(dbc, NULL, "DSN=MYDSN",
                 SQL_NTS, outstr, sizeof(outstr), &outstrlen,
                 SQL_DRIVER_COMPLETE);
```

The Easysoft ODBC-SQL Server Driver data source specified in the `SQLDriverConnect` call needs to connect with a Windows user name and password. For example:

```
[MYDSN]
Driver    = Easysoft ODBC-SQL Server Driver
Server    = myserver\SQLEXPRESS
User      = mywindowsuser
Password  = mywindowspassword
```

SQL_COPT_SS_PRESERVE_CURSORS

Whether the Easysoft ODBC-SQL Server Driver preserves cursors when `SQLEndTran` commits or rolls back a transaction.

(You can also configure this behaviour by using the `PreserveCursor` data source attribute. For more information, see **"Attribute Fields" on page 86**.)

Value	Description
SQL_PC_OFF	Default. Cursors are closed when a transaction is committed or rolled back by using `SQLEndTran`.
SQL_PC_ON	Cursors are preserved when a transaction is committed or rolled back by using `SQLEndTran`.

This C code sample uses `SQL_COPT_SS_PRESERVE_CURSORS` to preserve a cursor following a positioned update:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

main() {
  SQLHENV env;
  SQLHDBC dbc;
  SQLHSTMT stmt_select, stmt_update;
  SQLRETURN ret;
  SQLCHAR last_name[ 64 ], first_name[ 64 ], cursor_name[ 64 ], update_sql[ 64 ];
  SQLSMALLINT reports_to, cursor_len;
  SQLLEN indicator[ 3 ];

  /* Allocate an environment handle */
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
  /* We want ODBC 3 support */
  SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
                (void *) SQL_OV_ODBC3, 0);
  /* Allocate a connection handle */
  SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
```

```
/* Enable manual-commit mode */
SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
                  SQL_AUTOCOMMIT_OFF, 0);
/* Preserve cursors when transactions are committed/rolled */
/* back. Alternatively, add PreserveCursor = Yes to the DSN */
SQLSetConnectAttr(dbc, SQL_COPT_SS_PRESERVE_CURSORS,
                  (void *) SQL_PC_ON, 0);

/* Connect to Northwind through the sample DSN */
SQLDriverConnect(dbc, NULL,
                 "DSN=SQLSERVER_SAMPLE;Database=Northwind",
                 SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);
/* Allocate the statement handles */
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt_select);
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt_update);

/* Create dynamic, updateable cursor for the positioned update */
SQLSetStmtAttr(stmt_select, SQL_ATTR_CURSOR_TYPE,
               (void *) SQL_CURSOR_DYNAMIC, 0);
SQLSetStmtAttr(stmt_select, SQL_ATTR_CONCURRENCY,
               (void *) SQL_CONCUR_ROWVER, 0);
```

```
SQLExecDirect(stmt_select,
                "SELECT LastName, FirstName, ReportsTo FROM Employees FOR UPDATE",
               SQL_NTS);

SQLBindCol(stmt_select, 1, SQL_C_CHAR, last_name,
           sizeof(last_name), &indicator[ 0 ]);
SQLBindCol(stmt_select, 2, SQL_C_CHAR, first_name,
           sizeof(first_name), &indicator[ 1 ]);
SQLBindCol(stmt_select, 3, SQL_INTEGER, &reports_to,
           0, &indicator[ 2 ]);

/* Get the cursor name for use in the update statement  */
SQLGetCursorName(stmt_select, cursor_name,
                 sizeof(cursor_name), &cursor_len);

/* Move through the result set until the cursor is positioned */
/* on the row for Robert King */
do

  ret = SQLFetch(stmt_select);

while ((ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) &&
      (strcmp(first_name, "Robert") != 0 && strcmp(last_name, "King") != 0));
```

```
  /* Positioned update of Robert King's line manager */
  sprintf(update_sql,
          "UPDATE Employees SET ReportsTo = 2 WHERE CURRENT OF %s",
          cursor_name);

  SQLExecDirect(stmt_update, update_sql, SQL_NTS);
  /* Commit the transaction */
  SQLEndTran(SQL_HANDLE_DBC, dbc, SQL_COMMIT);

 /* The cursor is still open, because SQL_COPT_SS_PRESERVE_CURSORS is set to */
 /* SQL_PC_ON. Reposition the cursor and fetch the updated record. */
  SQLFetchScroll(stmt_select, SQL_FETCH_PRIOR, 0 );
  SQLFetch(stmt_select);

  /* Display updated record */
  printf("%s %s reports to employee ID: %ld\n", first_name,
         last_name, reports_to);

  SQLCloseCursor(stmt_update);  /* Close cursor */
  SQLDisconnect(dbc);           /* Disconnect from driver */
    SQLFreeHandle(SQL_HANDLE_DBC, dbc);
  SQLFreeHandle(SQL_HANDLE_ENV, env);
}
```

SQL_COPT_SS_TXN_ISOLATION

Sets the SQL Server 2005 or later snapshot isolation attribute..

Value	Description
SQL_TXN_SS_SNAPSHOT	Indicates that from one transaction you cannot see changes made in other transactions and that you cannot see changes even when requerying.

For more information about snapshot isolation, see **"Snapshot Isolation" on page 151**.

THE SQLSETSTMTATTR FUNCTION

The Easysoft ODBC-SQL Server Driver supports the following driver-specific statement attributes:

SQL_SOPT_SS_DEFER_PREPARE

Whether the Easysoft ODBC-SQL Server Driver defers query preparation until execution time.

The attribute is only relevant to SQLServer 2000 and later.

Value	Description
SQL_DP_ON	Default. After calling `SQLPrepare`, the Easysoft ODBC-SQL Server Driver defers statement preparation until `SQLExecute` or `SQLDescribeCol` is executed. Any errors in the statement are not known until these functions are executed.
SQL_DP_OFF	The Easysoft ODBC-SQL Server Driver prepares the statement as soon as `SQLPrepare` is executed. Any errors in the statement will cause the prepare to fail.

In this C code sample, deferred statement preparation is disabled. The invalid SQL statement the sample contains therefore fails as soon as `SQLPrepare` is called.

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

main() {

  SQLHENV env;
  SQLHDBC dbc;
  SQLHSTMT stmt;
  SQLRETURN ret;

  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
  SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
                (void *) SQL_OV_ODBC3, 0);
  SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);

  SQLDriverConnect(dbc, NULL,
                   "DSN=SQLSERVER_SAMPLE",
                   SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);
```

```
  /* Allocate the statement handle */
  SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
  /* Do not defer query preparation. Prepare the statement */
  /* as soon as SQLPrepare is executed */
  SQLSetStmtAttr(stmt, SQL_SOPT_SS_DEFER_PREPARE,
                 (SQLPOINTER) SQL_DP_OFF, 0);
  /* Invalid statement */
  ret = SQLPrepare(stmt, "select * from non_existent_table",
                   SQL_NTS);
  if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
    ret = SQLExecute(stmt);
    if (ret != SQL_SUCCESS || ret != SQL_SUCCESS_WITH_INFO) {
     /* Because the invalid statement was prepared immediately, */
     /* SQLPrepare (below) rather than SQLExecute returns the error. */
      extract_error("SQLExecute", stmt, SQL_HANDLE_STMT);
    }
  }  else {
    /* The statement is invalid and so cannot be prepared. */
    /* See "ODBC from C Tutorial Part 1", on the Easysoft web */
    /* site for a definition of extract_error(). */
    extract_error("SQLPrepare", stmt, SQL_HANDLE_STMT);
  }
}
```

Note that if the statement contains parameters, `SQLPrepare` returns `SQL_SUCCESS` even if the statement is invalid. Any errors in the statement are not known until the statement is executed or `SQLDescribeParam` is called. This behaviour happens regardless of how `SQL_SOPT_SS_DEFER_PREPARE` is set.

For example, in the following code extract, `SQLPrepare` succeeds even though the parameterised statement is invalid:

```
/* Do not defer query preparation. */
SQLSetStmtAttr(stmt, SQL_SOPT_SS_DEFER_PREPARE,
               (SQLPOINTER) SQL_DP_OFF, 0);

/* This statement is invalid. The parameter marker for the */
/* status column is missing. However, SQLPrepare still succeeds */
ret = SQLPrepare(stmt, "INSERT INTO Orders (OrderId, CustId,
OpenDate, SalesPerson, Status) VALUES (?, ?, ?, ?)", SQL_NTS);

if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
  SQLBindParameter();
  .
  .
  .
  /* The errors in the statement are not known until this point */
  SQLExecute(stmt);
}
```

Unicode Support

The Easysoft ODBC-SQL Server Driver is a Unicode driver that supports the Unicode version (with suffix "W") of the ODBC calls it implements. Using a Unicode driver with a Unicode application removes the need for the driver manager to map Unicode functions and data types to ANSI. This results in better performance and removes the restrictions inherent in the Unicode to ANSI mappings.

The Easysoft ODBC-SQL Server Driver supports the following SQL Server Unicode data types:

- `nchar`
- `ntext`
- `nvarchar`
- `nvarchar(max)`

Note The `nvarchar(max)` data type was introduced in SQL Server 2005.

ANSI-Only Version of the Easysoft ODBC-SQL Server Driver

The Easysoft ODBC-SQL Server Driver distribution includes an ANSI-only version of the driver that does not support the Unicode ODBC APIs. This version of the driver should not normally be needed and is only provided for use with old and non-conformant Driver Managers.

If you do need to use the ANSI-only driver, first install the driver under unixODBC. To do this, open `/etc/odbcinst.ini` in a text editor. Copy the section for the standard driver and paste it below the existing section. Change the `[driver name]` in the new section. In the `Driver` entry, suffix the library name with `_a` . For example:

```
[Easysoft ODBC-SQL Server]
Driver       = /usr/local/easysoft/sqlserver/lib/libessqlsrv.so
Setup        = /usr/local/easysoft/sqlserver/lib/libessqlsrvS.so
Threading    = 0
FileUsage    = 1
DontDLClose  = 1
UsageCount   = 1
# Install the ANSI driver by adding a new odbcinst.ini section.
# This example odbcinst.ini extract is from a Linux installation
# of the Easysoft ODBC-SQL Server Driver.
[Easysoft ODBC-SQL Server ANSI APIs]
Driver       = /usr/local/easysoft/sqlserver/lib/libessqlsrv_a.so
Setup        = /usr/local/easysoft/sqlserver/lib/libessqlsrvS.so
Threading    = 0
FileUsage    = 1
DontDLClose  = 1
UsageCount   = 1
```

In your data source, specify the new driver name in the `Driver` entry. For example:

```
[SQLSERVER_SAMPLE]
Driver  = Easysoft ODBC-SQL Server ANSI APIs
.
.
.
```

The xml Data Type

SQL Server 2005 introduced an `xml` data type for storing XML documents in table columns or Transact-SQL variables.

The Easysoft ODBC-SQL Server Driver supports the `xml` data type and its associated methods: `query()`, `value()`, `exist()`, `modify()` and `nodes()`.

The `query()` method lets you use an XML Query (XQuery) definition to search XML data stored in columns and variables of the `xml` type. The XQuery language is a World Wide Web Consortium (W3C) standard for retrieving or defining a set of XML nodes that meet a set of criteria.

In the following example, an XQuery is specified against the Instructions column in the ProductModel table. The Instructions column data type is `xml` and therefore exposes the `query()` method. The ProductModel table is contained in the SQL Server sample database AdventureWorks.

```
SELECT Instructions.query('declare namespace
AWMI="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelManuInstructions";
    /AWMI:root/AWMI:Location[@LocationID=10]
') as Result
FROM  Production.ProductModel
WHERE ProductModelID=7
```

The XQuery includes a namespace declaration, `declare namespace AWMI=...`, and a query expression, `/AWMI:root/AWMI:Location[@LocationID=10]`. The namespace declaration identifies the XML namespace associated with elements in the Instructions column. The query expression retrieves only those records for which the `LocationID` attribute value is 10:

```
<AWMI:Location
xmlns:AWMI="http://schemas.microsoft.com/sqlserver/2004/07/adventu
re-works/ProductModelManuInstructions"
LaborHours="2.5"...LocationID="10">

...
```

This second example uses the `query()` method to construct an XML element named `<Product>`. The `<Product>` element has a `ProductModelID` attribute, in which the `ProductModelID` attribute value is retrieved from the database.

```
SELECT CatalogDescription.query(' declare namespace
PD="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription"; <Product ProductModelID="{
/PD:ProductDescription[1]/@ProductModelID }" /> ') as Result

FROM Production.ProductModel
```

The `exist()` method lets you filter XML data. For example, add the following WHERE clause to the previous query to find only records that contain a `<Warranty>` element.

```
where CatalogDescription.exist(' declare namespace
PD="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription"; declare namespace
wm="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelWarrAndMain";
/PD:ProductDescription/PD:Features/wm:Warranty ') = 1
```

Note

When querying or updating `xml` columns or variables with the `xml` data type methods, the data source attributes `AnsiNPW and QuotedId` must be set to `Yes` (the default value for these settings). Otherwise, queries and modifications will fail for `xml` data types.

Using Large-Value Data Types

SQL Server 2005 introduced the max specifier, which expands the storage capabilities of the `varchar`, `nvarchar`, and `varbinary` data types to allow storage of values as large as 2 gigabytes (GB). `varchar(max)`, `nvarchar(max)`, and `varbinary(max)` are collectively called large-value data types.

The Easysoft ODBC-SQL Server Driver exposes the `varchar(max)`, `varbinary(max)` and `nvarchar(max)` types as `SQL_VARCHAR`, `SQL_VARBINARY`, and `SQL_WVARCHAR` in ODBC API functions that accept or return ODBC SQL data types.

When reporting the maximum size of a column, the Easysoft ODBC-SQL Server Driver will report either:

- The defined maximum size, which for example, is 2000 for a `varchar(2000)` column.

– OR –

- The value `SQL_SS_LENGTH_UNLIMITED` (0) `for varchar(max)`, `varbinary(max)` and `nvarchar(max)` columns.

Snapshot Isolation

SQL Server 2005 introduced a new transaction isolation level: snapshot. A snapshot transaction does not block updates executed by another transaction and can continue to read (but not update) the version of the data that existed when it started. Snapshot isolation is also called row versioning because SQL Server keeps "versions" of rows that are being changed: the original version and the version being changed.

Snapshot isolation is enabled for a database when the `ALLOW_SNAPSHOT_ISOLATION` database option is set to ON. For example, to enable snapshot isolation for the pubs sample database:

```
ALTER DATABASE pubs SET ALLOW_SNAPSHOT_ISOLATION ON
```

By default, this database option is set to OFF.

The Easysoft ODBC-SQL Server Driver supports snapshot isolation through the `SQLSetConnectAttr` and `SQLGetInfo` ODBC API functions.

For snapshot transactions, ODBC applications need to call `SQLSetConnectAttr` and set the `SQL_COPT_SS_TXN_ISOLATION` attribute to `SQL_TXN_SS_SNAPSHOT`. `SQL_TXN_SS_SNAPSHOT` indicates that the transaction will take place under the snapshot isolation level. For example:

```
SQLSetConnectAttr(dbc, SQL_COPT_SS_TXN_ISOLATION, (SQLPOINTER *)SQL_TXN_SS_SNAPSHOT, 0);
```

The `SQLGetInfo` function supports the `SQL_TXN_SS_SNAPSHOT` value, which has been added to the `SQL_TXN_ISOLATION_OPTION` info type.

The `SQL_COPT_SS_TXN_ISOLATION` and `SQL_TXN_SS_SNAPSHOT` attributes are Easysoft ODBC-SQL Server Driver Driver-specific ODBC extensions. To use these attributes, ODBC applications need to include the `sqlncli.h` header file. `sqlncli.h` is installed in `/usr/local/easysoft/sqlserver/include`.

Performing Bulk Copy Operations

BCP UTILITY

The `bcp` utility can be used to import large numbers of new rows into SQL Server tables or to export data out of tables into data files. Except when used with the `queryout` option, the utility requires no knowledge of SQL.

Syntax

```
bcp {[[database_name.][schema].]{table_name | view_name} | "query"}
    {in | out | queryout | format} data_file
    [-m max_errors] [-f format_file] [-x] [-D logfile]
    [-F first_row] [-L last_row] [-b batch_size]
    [-n] [-c] [-N] [-w] [-V (70 | 80 | 90 | 100 | 110 | 120)]
    [-q] [-C { RAW } ] [-t field_terminator]
    [-r row_terminator] [-o output_file] [-a packet_size]
    [-d database] [-e error_file] [-S server_name[\instance_name][:port]]
    [-U login] [-P password] [-useNTLMv2] [-A APP_NAME]
    [-T] [-v] [-k] [-g timeout] [-K] [-E] [-h"hint [,...n]"] [-Xe filename]
    [-Xc cypher] [-G]
```

Arguments

database_name

The name of the database where *table_name* or *view_name* is located. The database can also be specified with the `-d` option. If no database is specified, the default database for *login* is used.

schema

The name of the schema to which *table_name* or *view_name* belongs.

table_name

The name of the destination table when importing data into SQL Server (`in`), and the source table when exporting data from SQL Server (`out`).

view_name

The name of the destination view when importing data into SQL Server (`in`), and the source view when exporting data from SQL Server (`out`).

"*query*"

An SQL query that returns a result set. If the query returns multiple result sets, such as a `SELECT` statement that specifies a `COMPUTE` clause, only the first result set is copied to the data file; subsequent result sets are ignored. If the database containing the target table or view is not the default for *login*, specify the database with the `-d` option.

in | out | queryout | format

The direction of the bulk copy:

- `in` copies data from a file into a database table or view.
- `out` copies from a database table or view to a file. If you specify an existing file, the file is overwritten. The user who is running `bcp` must have write permission to the directory where the file is located. When extracting data, note that `bcp` represents an empty string as a null character (\0) and a NULL as a null value.
- `queryout` must be specified only when bulk copying data from a query.

- `format` creates a format file based on the option specified (`-n`, `-c`, `-w`, or `-N`) and the table or view delimiters. When bulk copying data, the `bcp` command can refer to a format file, which saves you from re-entering format information interactively. The format option requires the `-f` option; creating an XML format file, also requires the `-x` option.

data_file

The full path to the data file. When data is bulk imported into SQL Server, the data file contains the data to be copied into the specified table or view. When data is bulk exported from SQL Server, the data file contains the data copied from the table or view.

For the `format` option, you must specify `nul` as the value of *data_file* (`format nul`).

-m *max_errors*

The maximum number of times that `bcp` can return SQL_ERROR when bulk copying data from a query before the `bcp` operation is cancelled.

-f *format_file*

The full path to a format file:

- If `-f` is used with the `format` option, *`format_file`* is created for the specified table or view. To create an XML format file, also specify the `-x` option.
- If used with the `in` or `out` option, `-f` requires an existing format file.

-x

Used with the `format` and `-f` *`format_file`* options, the `-x` option generates an XML-based format file instead of the default non-XML format file. For example, `bcp AdventureWorks.Sales.Currency format nul -f bcp.xml -x -c -U mydomain\\myuser -P mypassword -S mymachine\\sqlexpress`.

-D *logfile*

Enables `bcp` logging and specifies the log file where the logging information is written. This can be a very useful debugging aid but it should be remembered that logging will slow `bcp` down, so remember to disable logging when you have finished debugging. Ensure that the user who is `bcp` has write permission to the log file (and to the directory containing it).

-F *first_row*

The number of the first row to export from a table or import from a data file. *first_row* should be a value greater than 0 but less than or equal to the total number rows. The default is the first row of the file.

-L *last_row*

The number of the last row to export from a table or import from a data file. *last_row* should be a value greater than 0 but less than or equal to the number of the last row. The default is the last row of the file.

-b *batch_size*

The number of rows per batch when importing data. Each batch is imported and logged as a separate transaction that imports the whole batch before being committed. By default, all the rows in the data file are imported as one batch. To distribute the rows among multiple batches, specify a *batch_size* that is smaller than the number of rows in the data file. If the transaction for any batch fails:

- Insertions from the current batch are rolled back.
- No further batches are inserted.

Batches already imported by committed transactions are unaffected by a subsequent failure.

-n

Use native (database) data types when importing and exporting data. This option does not prompt for each field; it uses the native values.

-c

Use character format when importing and exporting data. Character format uses the character data format for all columns. This option does not prompt for each field; it uses CHAR as the storage type, no prefixes, \t (tab character) as the field separator, and \n (newline character) as the row terminator.

-N

Use Unicode character format when importing and exporting character data. Use native format when importing and exporting non-character data.

-w

Use Unicode character format when importing and exporting data. Use this option when importing and exporting non-ASCII data stored in NCHAR, NVARCHAR and NTEXT columns. This option does not prompt for each field; it uses NCHAR as the storage type, no prefixes, \t (tab character) as the field separator, and \n (newline character) as the row terminator.

-V (70 | 80 | 90 | 100 | 110)

The version of SQL Server that bcp is connecting to, where:

70 = SQL Server 7.0

80 = SQL Server 2000

90 = SQL Server 2005

100 = SQL Server 2008

110 = SQL Server 2012 and above

-q

Executes the `SET QUOTED_IDENTIFIERS ON` statement in the connection between the `bcp` and an instance of SQL Server.

-C { RAW }

Specifies the code page of the data in the data file.

Value	Description
RAW	No conversion from one code page to another occurs.

-t *field_terminator*

The field terminator. The default is \t (tab character).

-r *row_terminator*

The row terminator. The default is \n (newline character).

-o *output_file*

The name of a file that receives `bcp` output redirected from the command prompt. For example, if you specified `-o /tmp/bcp.txt` and exported some data, `/tmp/bcp.txt` would contain something similar to:

```
Starting copy...

105 rows successfully bulk-copied to host file.
Total received: 105

Network packet size (bytes): 4096
```

The contents of *output_file* are overwritten each time you successfully import or export data.

-a *packet_size*

The packet size in bytes that `bcp` will request when sending data to and receiving data from SQL Server. The specified packet size must be lower than 65536 bytes.

The default packet size is 4096 bytes. After a successful import or export, the `bcp` output shows the packet size used.

-d *database*

The name of the database where *table_name* or *view_name* is located. If no database is specified, the default database for *login* is used.

-e *error_file*

The name of the file to write errors to.

-S *server_name*[*instance_name*]

The SQL Server instance that you want to connect to. To connect to the default instance of SQL Server on a server, specify only *server_name*. To connect to a named instance of SQL Server, specify *server_name**instance_name*. To include the port that the SQL Server instance is listening on, specify *server_name*:*port*.

-U *login*

The SQL Server login name to use when connecting to SQL Server. If the SQL Server instance uses Windows Authentication, specify the Windows user name to use to authenticate the connection. (Include the `-T` argument if you specify a Windows user name.) If the SQL Server instance permits SQL Server Authentication, you can also specify a SQL Server user name.

-P *password*

The password for *login*. You do not have to specify a password on the command line. If you omit the -P argument from the command line, bcp will prompt you for one when you run the command.

-useNTLMv2

If you want to use NTLMv2 to authenticate a Windows user specified with `-U`, include `-useNTLMv2` in your bcp command. If NTLMv2 is enabled at your site and you omit the `-useNTLMv2`, argument, the bcp connection will fail with the error "Login failed. The login is from an untrusted domain and cannot be used with Windows authentication."

-A *APP_NAME*

The application name that bcp registers with SQL Server. This is returned by the SQL Server function `APP_NAME()`. The default application name is `BCPTOOL`. Use the `-a` argument to override the default name.

-T

Use Windows authentication to validate the connection. If this argument is specified, *login* must be a Windows user name.

-v

Reports the `bcp` version number.

-k

Empty columns retain a null value during an import, rather than have any default values for the columns inserted.

-g [*timeout*]

The number of milliseconds to wait for a TCP connection to the SQL Server machine to be established before returning to the bcp. For more information, see the `LogonTimeout` attribute in **"Configuration" on page 86**.

-G [*req_flags*]

Set the `GSSFlag` attribute via bcp. For more information, see the `GSSFlag` attribute in **"Configuration" on page 86**.

-K

Whether to access a SQL Server instance as a Kerberos service.

When you include this argument, the Easysoft ODBC-SQL Server Driver will attempt to obtain a service ticket for the following Service Principal Name (SPN):

`MSSQLSvc/`*`server`*

where:

server is the name or IP address of the SQL Server machine specified with the -S argument.

Do not supply a user name or password if you include the -K argument. The Kerberos application `kinit` must have already been used for authentication on the Easysoft ODBC-SQL Server Driver machine. For more information about `kinit` and accessing SQL Server as a Kerberos service, see the following Easysoft tutorial:

http://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html

-M

The Service Principal Name (SPN) for a SQL Server instance that has been registered as a Kerberos service. If the SPN contains an instance name, you need to include the `-M` argument along with the -K argument. For example:

```
bcp -S "mysqlservermachine\myinstance" -K -M
MSSQLSvc/mysqlservermachine:myinstance
```

-E

Use identity values in the imported data file for the identity column. If `-E` is not specified, identity values in the data file being imported are ignored, and SQL Server automatically assigns unique values based on the seed and increment values specified during table creation.

-h"*hint* [,...*n*]"

The hint or hints to be used during a bulk import of data into a table or view.

ORDER(column [ASC | DESC] [,...n])

The sort order of the data in the data file.

ROWS_PER_BATCH = *num*

Number of rows of data per batch. Used when `-b` is not specified, resulting in the entire data file being sent to the server as a single transaction. The server optimises the bulk load according to the value *num*. By default, `ROWS_PER_BATCH` is unknown.

KILOBYTES_PER_BATCH = *num*

Approximate number of kilobytes of data per batch. By default, `KILOBYTES_PER_BATCH` is unknown.

TABLOCK

A bulk update table-level lock is acquired for the duration of the bulk load operation; otherwise, a row-level lock is acquired.

CHECK_CONSTRAINTS

All constraints on the target table or view must be checked during the bulk-import operation. Without the `CHECK_CONSTRAINTS` hint, any `CHECK` and `FOREIGN KEY` constraints are ignored, and after the operation the constraint on the table is marked as not-trusted.

FIRE_TRIGGERS

Specified with the `in` argument, any insert triggers defined on the destination table will run during the bulk-copy operation. If `FIRE_TRIGGERS` is not specified, no insert triggers will run.

Remarks

- The Easysoft `bcp` client is located in *`installation_dir`*`/easysoft/sqlserver/bcp` where *installation_dir* is the Easysoft installation directory, by default `/usr/local`.
- SQL Server identifiers can include embedded spaces and quotation marks. If the identifier contains spaces, enclose the database, table, view or schema name with quotation marks. If the identifier contains quotation marks, enclose the database, table, view or schema name with double quotation marks and square brackets ("[ ]").

-Xe *filename*

Use this option to specify a source for random data for an SSL connection. Use *`filename`* to specify a randomness device. You only need to use this option if `bcp` cannot find a source for random data on your system.

-Xc *cypher*

For an SSL connection, request a different encryption or data integrity algorithm to the ones negotiated during the SSL handshake. Separate multiple cypher suites with a colon. For example:

```
-Xc AES:3DES
```

EASYSOFT ODBC-SQL SERVER DRIVER EXTENSIONS - BULK COPY FUNCTIONS

In this section:

To build a program that uses the bulk copy extension library (`esbcp`), use this compile line format:

```
cc -I/usr/local/easysoft/unixODBC/include
myprogram.c -
I/usr/local/easysoft/sqlserver/include -o
myprogram -L/usr/local/easysoft/unixODBC/lib/ -
L/usr/local/easysoft/sqlserver/lib/ -lodbc -
lodbcinst -lesbcp
```

If you did not install the Easysoft ODBC-SQL Server Driver under `/usr/local` amend the path accordingly.

bcp_batch

Commits all rows previously bulk copied from program variables and sent to SQL Server by `bcp_sendrow`.

Syntax

```
DBINT bcp_batch (HDBC
        hdbc);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

Returns

The number of rows saved after the last call to `bcp_batch`, or -1 in case of error.

Remarks

Bulk copy batches define transactions. When an application uses `bcp_bind` and `bcp_sendrow` to bulk copy rows from program variables to SQL Server tables, the rows are committed only when the program calls `bcp_batch` or `bcp_done`.

Examples

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

SQLHENV henv = SQL_NULL_HENV;
HDBC hdbc1 = SQL_NULL_HDBC;
```

```
DBINT       idRow = 5;
char*       pPart1 = "Text chunk 1.";
char*       pPart2 = "Text chunk 2.";
char*       pPart3 = "Text chunk 3.";
DBINT       cbAllParts;
DBINT       nRowsProcessed;

void Cleanup() {
   if (hdbc1 != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc1);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   }

   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

int main() {
   RETCODE retcode;

   /* Allocate the ODBC environment and save handle. */
   retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(Env) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Notify ODBC that this is an ODBC 3.0 app. */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
      Cleanup();
```

```
      return(9);
   }

   /* Allocate ODBC connection handle, set BCP mode, and connect. */
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLSetConnectAttr(hdbc1, SQL_COPT_SS_BCP, (void *)SQL_BCP_ON,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetConnectAttr(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLDriverConnect(hdbc1, NULL, "DSN=MYDSN;", SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_COMPLETE);
   if ( (retcode != SQL_SUCCESS) && (retcode != SQL_SUCCESS_WITH_INFO) ) {
      printf("SQLDriverConnect() Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Initialise the bulk copy. */
   retcode = bcp_init(hdbc1, "articles", NULL, NULL, DB_IN);
   if ( (retcode != SUCCEED) ) {
      printf("bcp_init(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }
```

```
   /* Bind program variables to table columns. */
   retcode = (bcp_bind(hdbc1, (LPCBYTE) &idRow, 0, SQL_VARLEN_DATA, NULL, 0, SQLINT4,
1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_bind(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   cbAllParts = strlen(pPart1) + strlen(pPart2) + strlen(pPart3);

   retcode = (bcp_bind(hdbc1, NULL, 0, cbAllParts, NULL, 0, SQLTEXT, 2));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_bind(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* Now send this row, with the text value broken into three chunks. */
   retcode = (bcp_sendrow(hdbc1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_sendrow(hdbc1) Failed.n\n");
      Cleanup();
      return(9);
   }
   cbAllParts=strlen( pPart1 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 1\n\n");
      Cleanup();
      return(9);
   }
```

```
   cbAllParts=strlen( pPart2 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart2));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 2\n\n");
      Cleanup();
      return(9);
   }

   cbAllParts=strlen( pPart3 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart3));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 3\n\n");
      Cleanup();
      return(9);
   }

   /* We're all done. */
   nRowsProcessed = bcp_batch(hdbc1);

   /* Cleanup */
   SQLDisconnect(hdbc1);
   SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

bcp_bind

Binds data from a program variable to a table column for bulk copy into SQL Server.

Syntax

```
RETCODE bcp_bind (
        HDBC hdbc,
        LPCBYTE pData,
        INT cbIndicator,
        DBINT cbData,
        LPCBYTE pTerm,
        INT cbTerm,
        INT eDataType,
        INT idxServerCol);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`pData`

Is a pointer to the data copied. If `eDataType` is `SQLTEXT`, `SQLNTEXT`, `SQLXML`, `SQLUDT`, `SQLCHARACTER`, `SQLVARCHAR`, `SQLVARBINARY`, `SQLBINARY`, `SQLNCHAR`, or `SQLIMAGE`, `pData` can be NULL. A NULL `pData` indicates that long data values will be sent to SQL Server in chunks using `bcp_moretext`. The user should only set `pData` to NULL if the column corresponding to the user bound field is a BLOB column otherwise `bcp_bind` will fail.

`cbIndicator`

Is the length, in bytes, of a length or null indicator for the column's data. Valid indicator length values are 0 (when using no indicator), 1, 2, 4, or 8.

`cbData`

Is the count of bytes of data in the program variable, not including the length of any length or null indicator or terminator.

`pTerm`

Is a pointer to the byte pattern, if any, that marks the end of this program variable. If there is no terminator for the variable, set `pTerm` to NULL.

`cbTerm`

Is the count of bytes present in the terminator for the program variable, if any. If there is no terminator for the variable, set cbTerm to 0.

`eDataType`

Is the C data type of the program variable. The data in the program variable is converted to the type of the database column. If this parameter is 0, no conversion is performed.

The `eDataType` parameter is enumerated by the SQL Server data type tokens in `/usr/local/easysoft/sqlserver/include/sqlncli.h`, not the ODBC C data type enumerators.

`idxServerCol`

The ordinal position of the column in the database table to which the data is copied. The first column in a table is column 1. The ordinal position of a column is reported by `SQLColumns`.

Returns

SUCCEED or FAIL.

Remarks

Use `bcp_bind` for a fast, efficient way to copy data from a program variable into a table in SQL Server.

Make a separate `bcp_bind` call for every column in the SQL Server table into which you want to copy. After the necessary `bcp_bind` calls have been made, then call `bcp_sendrow` to send a row of data from your program variables to SQL Server. Rebinding a column is not supported.

Whenever you want SQL Server to commit the rows already received, call `bcp_batch`.

When there are no more rows to be inserted, call `bcp_done`. Failure to do so results in an error.

If *`pData`* for a column is set to NULL because its value will be supplied by calls to `bcp_moretext`, any subsequent columns with `eDataType` set to `SQLTEXT`, `SQLNTEXT`, `SQLXML`, `SQLUDT`, `SQLCHARACTER`, `SQLVARCHAR`, `SQLVARBINARY`, `SQLBINARY`, `SQLNCHAR`, or `SQLIMAGE` must also be bound with `pData` set to NULL, and their values must also be supplied by calls to `bcp_moretext`.

For new large value types, such as `varchar(max)`, `varbinary(max)`, or `nvarchar(max)`, you can use `SQLCHARACTER`, `SQLVARCHAR`, `SQLVARBINARY`, `SQLBINARY`, and `SQLNCHAR` as type indicators in the `eDataType` parameter.

bcp_colfmt

Specifies the source or target format of the data in a user file. When used as a source format, `bcp_colfmt` specifies the format of an existing data file used as the source of data in a bulk copy to a SQL Server table. When used as a target format, the data file is created using the column formats specified with `bcp_colfmt`.

Syntax

```
RETCODE bcp_colfmt (
        HDBC hdbc,
        INT idxUserDataCol,
        BYTE eUserDataType,
        INT cbIndicator,
        DBINT cbUserData,
        LPCBYTE pUserDataTerm,
        INT cbUserDataTerm,
        INT idxServerCol);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`idxUserDataCol`

Is the ordinal column number in the user data file for which the format is being specified. The first column is 1.

`eUserDataType`

Is the data type of this column in the user file.

The `eUserDataType` parameter is enumerated by the SQL Server data type tokens in `/usr/local/easysoft/sqlserver/include/sqlncli.h`, not the ODBC C data type enumerators. For example, you can specify a character string, ODBC type `SQL_C_CHAR`, using the SQL Server-specific type `SQLCHARACTER`.

To specify the default data representation for the SQL Server data type, set this parameter to 0.

`cbIndicator`

Is the length, in bytes, of a length/null indicator within the column data. Valid indicator length values are 0 (when using no indicator), 1, 2, 4, or 8.

`cbUserData`

Is the maximum length, in bytes, of this column's data in the user file, not including the length of any length indicator or terminator.

The `cbUserData` value represents the count of bytes of data. If character data is represented by Unicode wide characters, a positive `cbUserData` parameter value represents the number of characters multiplied by the size, in bytes, of each character.

`pUserDataTerm`

Is the terminator sequence to be used for this column. This parameter is useful mainly for character data types because all other types are of fixed length or, in the case of binary data, require an indicator of length to accurately record the number of bytes present.

`cbUserDataTerm`

Is the length, in bytes, of the terminator sequence to be used for this column. If no terminator is present or desired in the data, set this value to 0.

`idxServerCol`

Is the ordinal position of the column in the database table. The first column number is 1. The ordinal position of a column is reported by `SQLColumns`.

If this value is 0, bulk copy ignores the column in the data file.

Returns

SUCCEED or FAIL.

Remarks

The `bcp_colfmt` function allows you to specify the user-file format for bulk copies.

The `bcp_columns` function must be called before any calls to `bcp_colfmt`.

You must call `bcp_colfmt` once for each column in the user file.

You do not need to copy all data in a user file to the SQL Server table. To skip a column, specify the format of the data for the column, setting the `idxServerCol` parameter to 0. If you want to skip a column, you must specify its type.

bcp_collen

Sets the data length in the program variable for the current bulk copy into SQL Server.

Syntax

```
RETCODE bcp_collen (
        HDBC hdbc,
        DBINT cbData,
        INT idxServerCol);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`cbData`

Is the length of the data in the program variable, not including the length of any length indicator or terminator. Setting `cbData` to `SQL_NULL_DATA` indicates all rows copied to the server contain a NULL value for the column. Setting it to `SQL_VARLEN_DATA` indicates that a string terminator or other method is used to determine the length of data copied.

`idxServerCol`

Is the ordinal position of the column in the table to which the data is copied. The first column is 1.

Returns

SUCCEED or FAIL.

Remarks

The `bcp_collen` function allows you to change the data length in the program variable for a particular column when copying data to SQL Server with `bcp_sendrow`.

Initially, the data length is determined when `bcp_bind` is called. If the data length changes between calls to `bcp_sendrow` and no length prefix or terminator is being used, you can call `bcp_collen` to reset the length. The next call to `bcp_sendrow` uses the length set by the call to `bcp_collen`.

You must call `bcp_collen` once for each column in the table whose data length you want to modify.

Example

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

SQLHENV henv = SQL_NULL_HENV;
HDBC hdbc1 = SQL_NULL_HDBC;

DBINT      idRow = 5;
char*      pPart1 = "Text chunk 1.";
char*      pPart2 = "Text chunk 2.";
char*      pPart3 = "Text chunk 3.";
DBINT      cbAllParts;
DBINT      nRowsProcessed;

void Cleanup() {
   if (hdbc1 != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc1);
```

```
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   }

   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

int main() {
   RETCODE retcode;

   /* Allocate the ODBC environment and save handle. */
   retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(Env) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Notify ODBC that this is an ODBC 3.0 app. */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Allocate ODBC connection handle, set BCP mode, and connect. */
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }
```

```
   retcode = SQLSetConnectAttr(hdbc1, SQL_COPT_SS_BCP, (void *)SQL_BCP_ON,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetConnectAttr(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLDriverConnect(hdbc1, NULL, "DSN=MYDSN;", SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_COMPLETE);
   if ( (retcode != SQL_SUCCESS) && (retcode != SQL_SUCCESS_WITH_INFO) ) {
      printf("SQLDriverConnect() Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Initialise the bulk copy. */
   retcode = bcp_init(hdbc1, "articles", NULL, NULL, DB_IN);
   if ( (retcode != SUCCEED) ) {
      printf("bcp_init(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* Bind program variables to table columns. */
   retcode = (bcp_bind(hdbc1, (LPCBYTE) &idRow, 0, SQL_VARLEN_DATA, NULL, 0, SQLINT4,
1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_bind(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }
```

```
cbAllParts = strlen(pPart1) + strlen(pPart2) + strlen(pPart3);

retcode = (bcp_bind(hdbc1, NULL, 0, cbAllParts, NULL, 0, SQLTEXT, 2));
if ( (retcode != SUCCEED) ) {
   printf("bcp_bind(hdbc1) Failed.\n\n");
   Cleanup();
   return(9);
}

retcode = (bcp_collen(hdbc1, 100, 1));
if ( (retcode != SUCCEED) ) {
   printf("bcp_collen(hdbc1) Failed.\n\n");
   Cleanup();
   return(9);
}

retcode = (bcp_colptr(hdbc1, NULL , 2));
if ( (retcode != SUCCEED) ) {
   printf("bcp_colptr(hdbc1) Failed.\n\n");
   Cleanup();
   return(9);
}

/* Now send this row, with the text value broken into three chunks. */
retcode = (bcp_sendrow(hdbc1));
if ( (retcode != SUCCEED) ) {
   printf("bcp_sendrow(hdbc1) Failed.n\n");
   Cleanup();
   return(9);
}

cbAllParts=strlen( pPart1 );

retcode = (bcp_moretext(hdbc1, cbAllParts, pPart1));
```

```
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 1\n\n");
      Cleanup();
      return(9);
   }
   cbAllParts=strlen( pPart2 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart2));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 2\n\n");
      Cleanup();
      return(9);
   }

   cbAllParts=strlen( pPart3 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart3));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 3\n\n");
      Cleanup();
      return(9);
   }

   /* We're all done. */
   nRowsProcessed = bcp_done(hdbc1);

   /* Cleanup */
   SQLDisconnect(hdbc1);
   SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

bcp_colptr

Sets the program variable data address for the current copy into SQL Server.

Syntax

```
RETCODE bcp_colptr (
        HDBC hdbc,
        LPCBYTE pData,
        INT idxServerCol);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`pData`

Is a pointer to the data to copy. If the bound data type is large value type (such as `SQLTEXT` or `SQLIMAGE`), `pData` can be NULL. A NULL `pData` indicates long data values will be sent to SQL Server in chunks using `bcp_moretext`.

If `pData` is set to NULL and the column corresponding to the bound field is not a large value type, `bcp_colptr` fails.

`idxServerCol`

Is the ordinal position of the column in the database table to which the data is copied. The first column in a table is column 1. The ordinal position of a column is reported by `SQLColumns`.

Returns

SUCCEED or FAIL.

Remarks

The `bcp_colptr` function allows you to change the address of source data for a particular column when copying data to SQL Server with `bcp_sendrow`.

Initially, the pointer to user data is set by a call to `bcp_bind`. If the program variable data address changes between calls to `bcp_sendrow`, you can call `bcp_colptr` to reset the pointer to the data. The next call to bcp_sendrow sends the data addressed by the call to `bcp_colptr`.

There must be a separate `bcp_colptr` call for every column in the table whose data address you want to modify.

bcp_columns

Sets the total number of columns found in the user file for use with a bulk copy into or out of SQL Server. `bcp_setbulkmode` can be used instead of `bcp_columns` and `bcp_colfmt`.

Syntax

```
RETCODE bcp_columns (
        HDBC hdbc,
        INT nColumns);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`nColumns`

Is the total number of columns in the user file. Even if you are preparing to bulk copy data from the user file to an SQL Server table and do not intend to copy all columns in the user file, you must still set `nColumns` to the total number of user-file columns.

Returns

SUCCEED or FAIL.

Remarks

This function can be called only after `bcp_init` has been called with a valid file name.

You should call this function only if you intend to use a user-file format that differs from the default. For more information about a description of the default user-file format, see `bcp_init`.

After calling `bcp_columns`, you must call `bcp_colfmt` for each column in the user file to completely define a custom file format.

bcp_control

Changes the default settings for various control parameters for a bulk copy between a file and SQL Server.

Syntax

```
RETCODE bcp_control (
        HDBC hdbc,
        INT eOption,
        void* iValue);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`eOption`

Is one of the following:

BCPABORT

Stops a bulk-copy operation that is already in progress.

BCPBATCH

Is the number of rows per batch.

BCPDELAYREADFMT

If set to true, will cause `bcp_readfmt` to read at execution.

BCPFILEFMT

The version number of the data file format.

BCPFIRST

Is the first row of data to file or table to copy.

BCPFIRSTEX

For BCP out operations, specifies the first row of the database table to copy into the data file.

For BCP in operations, specifies the first row of the data file to copy into the database table.

BCPFMTXML

Specifies that the format file generated should be in XML format. It is off by default.

BCPHINTS

SQL Server bulk-copy processing hints or a Transact-SQL statement that returns a result set.

BCPKEEPIDENTITY

When `iValue` is TRUE, specifies that bulk copy functions insert data values supplied for SQL Server columns defined with an identity constraint. The input file must supply values for the identity columns. If this is not set, new identity values are generated for the inserted rows. Any data present in the file for the identity columns is ignored.

BCPKEEPNULLS

Specifies whether empty data values in the file will be converted to NULL values in the SQL Server table. When `iValue` is TRUE, empty values will be converted to NULL in the SQL Server table. The default is for empty values to be converted to a default value for the column in the SQL Server table if a default exists.

BCPLAST

Is the last row to copy.

BCPLASTEX

For BCP out operations, specifies the last row of the database table to copy into the data file.

For BCP in operations, specifies the last row of the data file to copy into the database table.

BCPMAXERRS

Is the number of errors allowed before the bulk copy operation fails.

BCPODBC

When TRUE, specifies that `datetime` and `smalldatetime` values saved in character format will use the ODBC timestamp escape sequence prefix and suffix. The BCPODBC option only applies to DB_OUT.

When FALSE, a datetime value representing January 1, 1997 is converted to the character string: 1997-01-01 00:00:00.000. When TRUE, the same datetime value is represented as: {ts '1997-01-01 00:00:00.000'}.

BCPROWCOUNT

Returns the number of rows affected by the current (or last) BCP operation.

BCPTEXTFILE

When TRUE, specifies that the data file is a text file, rather than a binary file.

BCPUNICODEFILE

When TRUE, specifies the input file is a Unicode file.

Returns

SUCCEED or FAIL.

Remarks

This function sets various control parameters for bulk-copy operations.

Example

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

SQLHENV henv = SQL_NULL_HENV;
HDBC hdbc1 = SQL_NULL_HDBC;
HDBC      hdbc;
DBINT      nRowsProcessed;

void Cleanup() {
   if (hdbc1 != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc1);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   }

   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

int main() {
   RETCODE retcode;

   /* Allocate the ODBC environment and save handle. */
```

```
   retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(Env) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Notify ODBC that this is an ODBC 3.0 app. */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Allocate ODBC connection handle, set BCP mode, and connect. */
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLSetConnectAttr(hdbc1, SQL_COPT_SS_BCP, (void *)SQL_BCP_ON,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetConnectAttr(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLDriverConnect(hdbc1, NULL, "DSN=MYDSN;", SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_COMPLETE);
```

```
   if ( (retcode != SQL_SUCCESS) && (retcode != SQL_SUCCESS_WITH_INFO) ) {
      printf("SQLDriverConnect() Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Initialise the bulk copy. */
   /* Table definition: CREATE TABLE myTable(ColA varchar(25), ColB varchar(25),ColC
varchar(25)) */
   retcode = bcp_init(hdbc1, "myTable", "myTable.dat", "myErrors.log", DB_IN);
   if ( (retcode != SUCCEED) ) {
      printf("bcp_init(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* Set the number of rows per batch */
   retcode = (bcp_control(hdbc1, BCPBATCH, (void*) 1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_control(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* There are three columns */
   retcode = (bcp_columns(hdbc1, 3));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_columns(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   bcp_colfmt(hdbc1, 1, SQLCHARACTER, 2, 25, "\t", 1, 1);
   bcp_colfmt(hdbc1, 2, SQLCHARACTER, 2, 25, "\t", 1, 2);
```

```
    bcp_colfmt(hdbc1, 3, SQLCHARACTER, 2, 25, "", 1, 3);

    retcode = (bcp_writefmt(hdbc1, "myFmtFile.fmt"));
    if ( (retcode != SUCCEED) ) {
       printf("bcp_writefmt(hdbc1) Failed.\n\n");
       Cleanup();
       return(9);
    }

    retcode = (bcp_readfmt(hdbc1, "myFmtFile.fmt"));
    if ( (retcode != SUCCEED) ) {
       printf("bcp_readfmt(hdbc1) Failed.\n\n");
       Cleanup();
       return(9);
    }

    retcode = (bcp_exec(hdbc1, &nRowsProcessed));
    if ( (retcode != SUCCEED) ) {
       printf("bcp_exec(hdbc1) Failed. 1\n\n");
       Cleanup();
       return(9);
    }

    /* Cleanup */
    SQLDisconnect(hdbc1);
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
    SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

bcp_done

When `bcp_sendrow` is used to bulk copy rows from program variables into SQL Server tables, rows are committed only when the user calls `bcp_batch` or `bcp_done`.

Syntax

```
DBINT bcp_done (
    HDBC hdbc);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

Returns

The number of rows permanently saved after the last call to `bcp_batch` or -1 in case of error.

Remarks

Call `bcp_done` after the last call to `bcp_sendrow` or `bcp_moretext`. Failure to call `bcp_done` after copying all data results in errors.

bcp_exec

Executes a complete bulk copy of data between a database table and a user file.

Syntax

```
RETCODE bcp_exec (
        HDBC hdbc,
        LPDBINT pnRowsProcessed);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`pnRowsProcessed`

Is a pointer to a DBINT. The bcp_exec function fills this DBINT with the number of rows successfully copied. If `pnRowsProcessed` is NULL, it is ignored by `bcp_exec`.

Returns

SUCCEED, or FAIL. The bcp_exec function returns SUCCEED if all rows are copied. `bcp_exec` returns FAIL if a complete failure occurs Check the `pnRowsProcessed` parameter for the number of rows successfully copied.

Remarks

This function copies data from a user file to a database table or vice versa, depending on the value of the `eDirection` parameter in `bcp_init`.

Before calling `bcp_exec`, call `bcp_init` with a valid user file name. Failure to do so results in an error.

Example

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

main(int argc, char *argv[] )
{
      SQLCHAR szDSN[]="MyDSN";
      SQLCHAR szUser[]="myuser";
      SQLCHAR szPass[]="mypassword";
      char szTable[]="mytable";
      char szInFile[]="mydatafile";
      char szErrFile[]="mylog.log";

      DBINT      nRowsProcessed;

      SQLRETURN       ret;
      SQLHANDLE       hEnv;
      SQLHANDLE       hDbc;
      SQLHANDLE       hStmt;

      SQLINTEGER cbLen;
      SQLCHAR szResult[10];

      printf("\n");
      printf("Simple Input.\n");
      printf("--------------\n");
      printf("DSN  : %s\n",szDSN);

      ret = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hEnv );
      if ( ret != SQL_SUCCESS)
                              printf( "SQLAllocHandle failed" );
```

```
      ret = SQLSetEnvAttr( hEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
SQL_IS_UINTEGER );
      if ( ret != SQL_SUCCESS)
                              printf( "SQLSetEnvAttr conn failed" );

      ret = SQLAllocHandle( SQL_HANDLE_DBC, hEnv, &hDbc );
      if ( ret != SQL_SUCCESS)
                              printf( "SQLAllocHandle conn failed" );

      ret = SQLSetConnectAttr(hDbc, SQL_COPT_SS_BCP, (SQLPOINTER) SQL_BCP_ON,
SQL_IS_INTEGER);

      ret = SQLConnect( hDbc, szDSN, SQL_NTS, szUser, SQL_NTS, szPass, SQL_NTS);

      if ( SQL_SUCCEEDED( ret ))
      {
            printf("Connected successfully.\n");
      }
      else
      {
            printf("Connection failed.\n");
            return(1);
      }
     printf("bcp_init\n     Table  : %s\n     File   : %s\n     Errors : %s\n", szTable,
szInFile, szErrFile );

      ret = bcp_init(hDbc, szTable, szInFile, szErrFile, DB_IN);
      printf("bcp_init ret %d\n",ret);
      if (ret == FAIL )
      {
            printf("BCP bcp_init failed.\n");
            return(1);
      }
      nRowsProcessed=0;
```

```
        ret = bcp_exec(hDbc, &nRowsProcessed);
        printf("bcp_exec ret %d\n",ret);
        if (ret == FAIL )
        {
              if (nRowsProcessed == -1)
              {
                    printf("No rows processed\n");
              }
              else
              {
                    printf("Only %d rows processed\n",nRowsProcessed);
              }
              return(1);
        }
        printf("Rows : %d\n",nRowsProcessed);
        ret = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, &hStmt);

        SQLFreeHandle(SQL_HANDLE_STMT, hStmt);
        SQLFreeHandle(SQL_HANDLE_DBC, hDbc);
        SQLFreeHandle(SQL_HANDLE_ENV, hEnv);

        return SQL_SUCCESS;
}
```

bcp_getcolfmt

Used to find the column format property value.

Syntax

```
        HDBC hdbc,
        INT field,
        INT property,
        void* pValue,
        INT cbvalue,
        INT* pcbLen);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`field`

Is the column number for which the property is retrieved.

`property`

Is one of the property constants.

`pValue`

Is the pointer to the buffer from which to retrieve the property value.

`cbValue`

Is the length of the property buffer in bytes.

`pcbLen`

Pointer to length of the data that is being returned in the property buffer.

Returns

SUCCEED or FAIL.

Remarks

Column format property values are listed in the `bcp_setcolfmt` topic. The column format property values are set by calling the `bcp_setcolfmt` function, and the `bcp_getcolfmt` function is used to find the column format property value.

Example

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

SQLHENV henv = SQL_NULL_HENV;
HDBC hdbc1 = SQL_NULL_HDBC;
HDBC      hdbc;
DBINT      nRowsProcessed;

void Cleanup() {
   if (hdbc1 != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc1);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   }

   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

int main() {
   RETCODE retcode;
```

```
   /* Allocate the ODBC environment and save handle. */
   retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(Env) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Notify ODBC that this is an ODBC 3.0 app. */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Allocate ODBC connection handle, set BCP mode, and connect. */
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLSetConnectAttr(hdbc1, SQL_COPT_SS_BCP, (void *)SQL_BCP_ON,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetConnectAttr(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }
```

```
   retcode = SQLDriverConnect(hdbc1, NULL, "DSN=MYDSN;", SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_COMPLETE);
   if ( (retcode != SQL_SUCCESS) && (retcode != SQL_SUCCESS_WITH_INFO) ) {
      printf("SQLDriverConnect() Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Initialise the bulk copy. */
   /* Table definition: CREATE TABLE myTable(ColA varchar(25), ColB varchar(25),ColC
varchar(25)) */
   retcode = bcp_init(hdbc1, "myTable", "myTable.dat", "myErrors.log", DB_IN);
   if ( (retcode != SUCCEED) ) {
      printf("bcp_init(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* There are three columns */
   retcode = (bcp_columns(hdbc1, 3));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_columns(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   bcp_colfmt(hdbc1, 1, SQLCHARACTER, 2, 25, "\t", 1, 1);
   bcp_colfmt(hdbc1, 2, SQLCHARACTER, 2, 25, "\t", 1, 2);
   bcp_colfmt(hdbc1, 3, SQLCHARACTER, 2, 25, "", 1, 3);

   retcode = (bcp_writefmt(hdbc1, "myFmtFile.fmt"));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_writefmt(hdbc1) Failed.\n\n");
      Cleanup();
```

```
      return(9);
   }

   retcode = (bcp_getcolfmt(hdbc1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_getcolfmt(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   retcode = (bcp_exec(hdbc1, &nRowsProcessed));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_exec(hdbc1) Failed. 1\n\n");
      Cleanup();
      return(9);
   }

   /* Cleanup */
   SQLDisconnect(hdbc1);
   SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

bcp_gettypename

Returns the SQL type name for a specified BCP type token.

Syntax

```
RETCODE bcp_gettypename (
        INT token,
        DBBOOL fIsMaxType);
```

Arguments

`token`

A value indicating a BCP type token.

`field`

Indicates if token requested is a MAX type.

Returns

A string containing the SQL type name corresponding to the BCP type. If an invalid BCP type is specified, an empty string is returned.

Remarks

The BCP type tokens are defined in the `/usr/local/easysoft/sqlserver/include/sqlncli.h` header file.

bcp_init

Initialises the bulk copy operation.

Syntax

```
RETCODE bcp_init (
        HDBC hdbc,
        LPCTSTR szTable,
        LPCTSTR szDataFile,
        LPCTSTR szErrorFile,
        INT eDirection);
```

Unicode and ANSI names:

`bcp_initA` (ANSI)

`bcp_initW` (Unicode)

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`szTable`

Is the name of the database table to be copied into or out of. This name can also include the database name or the owner name. For example, `pubs.gracie.titles`, `pubs..titles`, `gracie.titles`, and titles are all valid table names.

If `eDirection` is DB_OUT, `szTable` can also be the name of a database view.

If `eDirection` is DB_OUT and a SELECT statement is specified using `bcp_control` before `bcp_exec` is called, `bcp_init szTable` must be set to NULL.

`szDataFile`

Is the name of the user file to be copied into or out of. If data is being copied directly from variables by using `bcp_sendrow`, set `szDataFile` to NULL.

`szErrorFile`

Is the name of the error file to be filled with progress messages, error messages, and copies of any rows that, for any reason, could not be copied from a user file to a table. If NULL is passed as `szErrorFile`, no error file is used.

`eDirection`

Is the direction of the copy, either DB_IN or DB_OUT. DB_IN indicates a copy from program variables or a user file to a table. DB_OUT indicates a copy from a database table to a user file. You must specify a user file name with DB_OUT.

Returns

SUCCEED or FAIL.

Remarks

Call `bcp_init` before calling any other bulk-copy function. `bcp_init` performs the necessary initialisations for a bulk copy of data between the Easysoft ODBC-SQL Server Driver and SQL Server.

The `bcp_init` function must be provided with an ODBC connection handle enabled for use with bulk copy functions. To enable the handle, use `SQLSetConnectAttr` with SQL_COPT_SS_BCP set to SQL_BCP_ON on an allocated, but not connected, connection handle. Attempting to assign the attribute on a connected handle results in an error.

When a data file is specified, `bcp_init` examines the structure of the database source or target table, not the data file. `bcp_init` specifies data format values for the data file based on each column in the database table, view, or SELECT result set. This specification includes the data type of each column, the presence or absence of a length or null indicator and terminator byte strings in the data, and the width of fixed-length data types. `bcp_init` sets these values as follows:

When copying to SQL Server, the data file must have data for each column in the database table. When copying from SQL Server, data from all columns in the database table, view, or SELECT result set are copied to the data file.

To change data format values specified for a data file, call `bcp_columns` and `bcp_colfmt`.

If no data file is used, you must call `bcp_bind` to specify the format and location in memory of the data for each column, then copy data rows to the SQL Server using `bcp_sendrow`.

bcp_moretext

Sends part of a long, variable-length data type value to SQL Server.

Syntax

```
RETCODE bcp_moretext (
        HDBC hdbc,
        DBINT cbData,
        LPCBYTE pData);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`cbData`

Is the number of bytes of data being copied to SQL Server from the data referenced by pData. A value of SQL_NULL_DATA indicates NULL.

`pData`

Is a pointer to the supported, long, variable-length data chunk to be sent to SQL Server.

Returns

SUCCEED or FAIL.

Remarks

This function can be used in conjunction with `bcp_bind` and `bcp_sendrow` to copy long, variable-length data values to SQL Server in a number of smaller chunks. `bcp_moretext` can be used with columns that have the following SQL Server data types: `text`, `ntext`, `image`, `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, `user-defined type (UDT)`, and `XML`. `bcp_moretext` does not support data conversions, the data supplied must match the data type of the target column.

If `bcp_bind` is called with a non-NULL pData parameter for data types that are supported by `bcp_moretext`, `bcp_sendrow` sends the entire data value, regardless of length. If, however, `bcp_bind` has a NULL pData parameter for supported data types, `bcp_moretext` can be used to copy data immediately after a successful return from `bcp_sendrow` indicating that any bound columns with data present have been processed.

bcp_readfmt

Reads a data file format definition from the specified format file.

Syntax

```
        HDBC hdbc,

        LPCTSTR szFormatFile);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`szFormatFile`

Is the path and file name of the file containing the format values for the data file.

Returns

SUCCEED or FAIL.

Remarks

After `bcp_readfmt` reads the format values, it makes the appropriate calls to `bcp_columns` and `bcp_colfmt`. There is no need for you to parse a format file and make these calls.

To persist a format file, call `bcp_writefmt`.

The bulk-copy utility (`bcp`) can also save user-defined data formats in files that can be referenced by `bcp_readfmt`.

bcp_sendrow

Sends a row of data to SQL Server.

Syntax

```
RETCODE bcp_sendrow (
    HDBC hdbc);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

Returns

SUCCEED or FAIL

Remarks

The `bcp_sendrow` function builds a row from variables bound with `bcp_bind` and sends it to SQL Server.

If `bcp_bind` is called specifying a long, variable-length data type, for example, an *eDataType* parameter of `SQLTEXT` and a non-NULL *pData* parameter, `bcp_sendrow` sends the entire data value. If, however, `bcp_bind` has a NULL *pData* parameter, `bcp_sendrow` returns control to the application immediately after all columns with data specified are sent to SQL Server. The application can then call `bcp_moretext` repeatedly to send the long, variable-length data to SQL Server, a chunk at a time. For more information, see `bcp_moretext`.

When `bcp_sendrow` is used to bulk copy rows from program variables into SQL Server tables, rows are committed only when the user calls `bcp_batch` or `bcp_done`. The user can choose to call `bcp_batch` once every n rows or when there is a lull between periods of incoming data. If `bcp_batch` is never called, the rows are committed when `bcp_done` is called.

Example

```
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include <sqlncli.h>

SQLHENV henv = SQL_NULL_HENV;
HDBC hdbc1 = SQL_NULL_HDBC;

DBINT      idRow = 5;
char*      pPart1 = "Text chunk 1.";
char*      pPart2 = "Text chunk 2.";
char*      pPart3 = "Text chunk 3.";
DBINT      cbAllParts;
DBINT      nRowsProcessed;
```

```
void Cleanup() {
   if (hdbc1 != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc1);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   }

   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

int main() {
   RETCODE retcode;

   /* Allocate the ODBC environment and save handle. */
   retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLAllocHandle(Env) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Notify ODBC that this is an ODBC 3.0 app. */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Allocate ODBC connection handle, set BCP mode, and connect. */
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
```

```
      printf("SQLAllocHandle(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLSetConnectAttr(hdbc1, SQL_COPT_SS_BCP, (void *)SQL_BCP_ON,
SQL_IS_INTEGER);
   if ( (retcode != SQL_SUCCESS_WITH_INFO) && (retcode != SQL_SUCCESS)) {
      printf("SQLSetConnectAttr(hdbc1) Failed\n\n");
      Cleanup();
      return(9);
   }

   retcode = SQLDriverConnect(hdbc1, NULL, "DSN=MYDSN;", SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_COMPLETE);
   if ( (retcode != SQL_SUCCESS) && (retcode != SQL_SUCCESS_WITH_INFO) ) {
      printf("SQLDriverConnect() Failed\n\n");
      Cleanup();
      return(9);
   }

   /* Initialise the bulk copy. */
   retcode = bcp_init(hdbc1, "MyTable", NULL, NULL, DB_IN);
   if ( (retcode != SUCCEED) ) {
      printf("bcp_init(hdbc1) Failed.\n\n");
      Cleanup();
      return(9);
   }

   /* Bind program variables to table columns. */
   retcode = (bcp_bind(hdbc1, (LPCBYTE) &idRow, 0, SQL_VARLEN_DATA, NULL, 0, SQLINT4,
1));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_bind(hdbc1) Failed.\n\n");
```

```
   Cleanup();
   return(9);
}

cbAllParts = strlen(pPart1) + strlen(pPart2) + strlen(pPart3);

retcode = (bcp_bind(hdbc1, NULL, 0, cbAllParts, NULL, 0, SQLTEXT, 2));
if ( (retcode != SUCCEED) ) {
   printf("bcp_bind(hdbc1) Failed.\n\n");
   Cleanup();
   return(9);
}

/* Now send this row, with the text value broken into three chunks. */
retcode = (bcp_sendrow(hdbc1));
if ( (retcode != SUCCEED) ) {
   printf("bcp_sendrow(hdbc1) Failed.n\n");
   Cleanup();
   return(9);
}
cbAllParts=strlen( pPart1 );

retcode = (bcp_moretext(hdbc1, cbAllParts, pPart1));
if ( (retcode != SUCCEED) ) {
   printf("bcp_moretext(hdbc1) Failed. 1\n\n");
   Cleanup();
   return(9);
}
cbAllParts=strlen( pPart2 );

retcode = (bcp_moretext(hdbc1, cbAllParts, pPart2));
if ( (retcode != SUCCEED) ) {
   printf("bcp_moretext(hdbc1) Failed. 2\n\n");
   Cleanup();
```

```
      return(9);
   }

   cbAllParts=strlen( pPart3 );

   retcode = (bcp_moretext(hdbc1, cbAllParts, pPart3));
   if ( (retcode != SUCCEED) ) {
      printf("bcp_moretext(hdbc1) Failed. 3\n\n");
      Cleanup();
      return(9);
   }

   /* We're all done. */
   nRowsProcessed = bcp_done(hdbc1);

   /* Cleanup */
   SQLDisconnect(hdbc1);
   SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
   SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

bcp_setbulkmode

`bcp_setbulkmode` lets you specify the column format in a bulk copy operation, setting all the column attributes in a single function call.

Syntax

```
RETCODE bcp_setbulkmode (
   HDBC hdbc,
   INT property,
   void * pField,
   INT cbField,
   void * pRow,
   INT cbRow
);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`property`

A constant of type BYTE. See the table in the Remarks section for a list of the constants.

`pField`

The pointer to the field terminator value.

`pRow`

The pointer to the row terminator value.

`cbRow`

The length in bytes of the row terminator value.

Returns

SUCCEED or FAIL

Remarks

`bcp_setbulkmode` can be used to bulk copy out of either a query or a table. When `bcp_setbulkmode` is used to bulk copy out a query statement, it must be called before calling `bcp_control` with BCP_HINT.

`bcp_setbulkmode` is an alternative to using `bcp_setcolfmt` and `bcp_columns`, which only let you specify the format of one column per function call.

bcp_setcolfmt

This function provides a flexible approach to specifying the column format in a bulk copy operation. It is used to set individual column format attributes. Each call to bcp_setcolfmt sets one column format attribute.

The `bcp_setcolfmt` function specifies the source or target format of the data in a user file. When used as a source format, `bcp_setcolfmt` specifies the format of an existing data file used as a data source of data in a bulk copy to a table in SQL Server. When used as a target format, the data file is created using the column formats specified with `bcp_setcolfmt`.

Syntax

```
RETCODE bcp_setcolfmt (
        HDBC hdbc,
        INT field,
        INT property,
        void* pValue,
        INT cbValue);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`field`

Is the ordinal column number for which the property is being set.

`property`

Is one of the property constants. Property constants are defined in this table.

Value	Description
BCP_FMT_TYPE	`BYTE` Is the data type of this column in the user file. If different from the data type of the corresponding column in the database table, bulk copy converts the data if possible. The BCP_FMT_TYPE parameter is enumerated by the SQL Server data type tokens in `/usr/local/easysoft/sqlserver/include/sqlncli.h`, rather than the ODBC C data type enumerators. For example, you can specify a character string, ODBC type SQL_C_CHAR, using the SQLCHARACTER type specific to SQL Server. To specify the default data representation for the SQL Server data type, set this parameter to 0.
BCP_FMT_DATA_LEN	`DBINT` Is the length in bytes of the data (column length).
BCP_FMT_TERMINATOR	`LPCBYTE` Pointer to the terminator sequence (either ANSI or Unicode as appropriate) to be used for this column. This parameter is useful mainly for character data types because all other types are of fixed length or, in the case of binary data, require an indicator of length to accurately record the number of bytes present.
BCP_FMT_SERVER_COL	`INT` Ordinal position of the column in the database
BCP_FMT_COLLATION	`LPCSTR` Collation name.

`pValue`

Is the pointer to the value to associate to the property. It allows each column format property to be set individually.

`cbvalue`

Is the length of the property buffer in bytes.

Returns

SUCCEED or FAIL.

Remarks

The `bcp_setcolfmt` function allows you to specify the user-file format for bulk copies.

You do not need to copy all data in a user file to the SQL Server table. To skip a column, specify the format of the data for the column, setting the BCP_FMT_SERVER_COL parameter to 0. If you want to skip a column, you must specify its type.

bcp_writefmt

Creates a format file containing a description of the format of the current bulk copy data file.

Syntax

```
RETCODE bcp_writefmt (
        HDBC hdbc,
        LPCTSTR szFormatFile);
```

Arguments

`hdbc`

Is the bulk copy-enabled ODBC connection handle.

`szFormatFile`

Is the path and file name of the user file to receive format values for the data file.

Returns

SUCCEED or FAIL.

Remarks

The format file specifies the data format of a data file created by bulk copy.

Table-Valued Parameters

Table-valued parameters, introduced in SQL Server 2008, allow multiple rows or values to be passed to a query or stored procedure in one call. Table-valued parameters decrease network latency by reducing network round trips. For example, given the task of updating multiple order line items an application traditionally would call one procedure to update the order and another procedure to update the line items. The second procedure would be called multiple times, once for each line item, therefore requiring multiple database round trips to fulfill the objective.

The Easysoft ODBC-SQL Server Driver enables data to be sent as a table-valued parameter with all values in memory.

The following example inserts an order and multiple order detail rows by passing a table-valued parameter to one stored procedure.

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <stdlib.h>
#include <string.h>
#include "sqlncli.h"

main() {
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT hstmt;
```

```
 /* ODBC API return status */
SQLRETURN ret;

/* SQL parameters */
#define ITEM_ARRAY_SIZE 20

SQLCHAR CustCode[6];
SQLCHAR *TVP = (SQLCHAR *) "TVParam";
SQLINTEGER ProdCode[ITEM_ARRAY_SIZE], Qty[ITEM_ARRAY_SIZE];
SQLINTEGER OrdNo;
char OrdDate[23];

/* Indicator/length variables associated with SQL parameters */
SQLLEN cbCustCode, cbTVP, cbProdCode[ITEM_ARRAY_SIZE],
cbQty[ITEM_ARRAY_SIZE], cbOrdNo, cbOrdDate;

/* Allocate an environment handle */
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);

/* We want ODBC 3 support */
SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3,
0);
```

```
/* Allocate a connection handle */
SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);

/* Connect to the DSN mydsn.
 * You will need to change mydsn to one you have created and tested
 * The SQL Server database your DSN connects to needs:
 *
 * These tables:
 *
 * CREATE TABLE dbo.TVPOrder (
 *   CustCode varchar(5),
 *   OrdNo int identity,
 *   OrdDate datetime
 *  )
 *
 * CREATE TABLE dbo.TVPItem (
 *   OrdNo int,
 *   ProdCode int,
 *   Qty int
 *  )
 *
 *
```

```
 * This user-defined table type:
 *
 * IF (SELECT COUNT(*) FROM sys.table_types
 * WHERE name = 'TVPParam' AND schema_id = 1) = 0
 * CREATE TYPE dbo.TVPParam AS TABLE(ProdCode integer, Qty
integer)
 *
 * This procedure:
 *
 * CREATE PROCEDURE
 * dbo.TVPOrderEntry
 * (
 *   @CustCode varchar(5),
 *   @Items TVPParam READONLY,
 *   @OrdNo integer output,
 *   @OrdDate datetime output)
 * AS
 *   SET @OrdDate = GETDATE();
 *
 *   INSERT INTO TVPOrder (OrdDate, CustCode)
 *     VALUES (@OrdDate, @CustCode);
 *
 *   SELECT @OrdNo = SCOPE_IDENTITY();
```

```
 *
 *    INSERT INTO TVPItem (OrdNo, ProdCode, Qty)
 *    SELECT @OrdNo, ProdCode, Qty FROM @Items
 */
SQLDriverConnect(dbc, NULL, "DSN=SQLSERVER_2012_EXPRESS_64_BIT;",
SQL_NTS,
                  NULL, 0, NULL, SQL_DRIVER_COMPLETE);

SQLAllocHandle(SQL_HANDLE_STMT, dbc, &hstmt);

/* Bind parameters for call to TVPOrderEntry */
/* 1 - CustCode input */
ret = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 5, 0, CustCode, sizeof(CustCode), &cbCustCode);

/* 2 - Items TVP */
ret = SQLBindParameter(hstmt,
    2,
    SQL_PARAM_INPUT,
    SQL_C_DEFAULT,
    SQL_SS_TABLE,
    ITEM_ARRAY_SIZE, /* ColumnSize: For a table-valued parameter
this is the row array size */
```

```
    0, /* DecimalDigits: For a table-valued parameter this is
always 0 */
   TVP, /* ParameterValuePtr: For a table-valued parameter this is
the type name of the */
         /* table-valued parameter, and also a token returned by
SQLParamData */
   strlen(TVP), /* BufferLength: For a table-valued parameter this
is the length of the type name or SQL_NTS */
   &cbTVP); /* StrLen_or_IndPtr: For a table-valued parameter this
is the number of rows actually used */


cbOrdNo = 0;
cbOrdDate = 0;


/* 3 - OrdNo output */
ret = SQLBindParameter(hstmt, 3, SQL_PARAM_OUTPUT,SQL_C_LONG,
SQL_INTEGER, 0, 0, &OrdNo,
   sizeof(SQLINTEGER), &cbOrdNo);


/* 4 - OrdDate output */
ret = SQLBindParameter(hstmt, 4, SQL_PARAM_OUTPUT,
SQL_C_CHAR,SQL_TYPE_TIMESTAMP, 23, 3, &OrdDate,
   sizeof(OrdDate), &cbOrdDate);


/* Bind columns for the table-valued parameter (parameter 2) */
```

```
/* First set focus on parameter 2  */
ret = SQLSetStmtAttr(hstmt, SQL_SOPT_SS_PARAM_FOCUS, (SQLPOINTER)
2, SQL_IS_INTEGER);

/* Col 1 - ProdCode */
 ret = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, ProdCode, sizeof(SQLINTEGER), cbProdCode);

/* Col 2 - Qty */
ret = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, Qty, sizeof(SQLINTEGER), cbQty);

/* Reset parameter focus */
ret = SQLSetStmtAttr(hstmt, SQL_SOPT_SS_PARAM_FOCUS, (SQLPOINTER)
0, SQL_IS_INTEGER);

/* Populate parameters */
cbTVP = 0; /* Number of rows available for input */

strcpy((char *) CustCode, "BEAUC"); cbCustCode = SQL_NTS;

ProdCode[cbTVP] = 555;cbProdCode[cbTVP] = sizeof(SQLINTEGER);
Qty[cbTVP] = 5;cbQty[cbTVP] = sizeof(SQLINTEGER);
cbTVP++; /* Number of rows available for input */
```

```
ProdCode[cbTVP] = 666;cbProdCode[cbTVP] = sizeof(SQLINTEGER);
Qty[cbTVP] = 6;cbQty[cbTVP] = sizeof(SQLINTEGER);

cbTVP++; /* Number of rows available for input */

/* Call the procedure */
ret = SQLExecDirect(hstmt, (SQLCHAR *) "{call TVPOrderEntry(?, ?,
?, ?)}",SQL_NTS);

}
```

Binding Procedure Parameters by Name

The Easysoft ODBC-SQL Server Driver supports named parameters, which allows an application to specify stored procedure parameters by name instead of by position in the procedure call.

This C code sample calls the AdventureWorks2008 stored procedure `uspSearchCandidateResumes`. The sample specifies the stored procedure's parameters by name.

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

main() {
   SQLHENV env;
   SQLHDBC dbc;
   SQLHSTMT stmt;
   SQLHANDLE ipd;
   SQLRETURN ret;

   /* Procedure input */
   SQLCHAR param_name_1 [ 64 ], param_name_2 [ 64 ],
           param_name_3 [ 64 ], param_name_4 [ 64 ];
   /* Search candidate resumes for this text: */
   SQLCHAR search_string[ 64 ] = "ISO9000";
```

```
/* Enable the default values for the other procedure */
/* parameters to be overridden. Initialise variables */
/* to the corresponding parameter's default value. */
BOOL use_inflectional = 0;
BOOL use_thesaurus = 0;
SQLSMALLINT language = 0;

/* Procedure output */
SQLCHAR col_1_name [ 64 ], col_2_name [ 64 ];
SQLINTEGER col_1;
SQLSMALLINT col_2, col_1_name_length, col_2_name_length;
SQLLEN len_1, len_2;

/* Allocate an environment handle */
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);

/* We want ODBC 3 support */
SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *)
              SQL_OV_ODBC3, 0);

/* Allocate a connection handle */
SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
```

```
/* Connect to the DSN mydsn */
/* Change mydsn to one you have created and tested */
SQLDriverConnect(dbc, NULL, "DSN=mydsn;", SQL_NTS,
                 NULL, 0, NULL, SQL_DRIVER_COMPLETE);

/* Allocate a statement handle */
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

/* Prepare the statement that will call the procedure */
SQLPrepare(stmt, "{call uspSearchCandidateResumes (?, ?, ?, ?)}",
           SQL_NTS);

/* Bind local variables to the parameters in the preceding */
/* statement. */
SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
                 SQL_CHAR, 1000, 0, search_string, 0, 0);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_BIT, SQL_BIT,
                 1, 0, &use_inflectional, 0, 0);
SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_BIT, SQL_BIT,
                 1, 0, &use_thesaurus, 0, 0);
SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_INTEGER,
                 SQL_INTEGER, 1, 0, &language, 0, 0);
```

```
/* Bind columns in the procedure result set to local */
/* variables */
SQLBindCol(stmt, 1, SQL_INTEGER, &col_1, 0, &len_1);
SQLBindCol(stmt, 2, SQL_INTEGER, &col_2, 0, &len_2);

/* Get IPD handle. The IPD contains information about the */
/* statement parameters, such as their SQL data types, */
/* lengths, and nullability. */
SQLGetStmtAttr(stmt, SQL_ATTR_IMP_PARAM_DESC, &ipd, 0, 0);

/* Set the SQL_DESC_NAME field of the IPD to the parameter */
/* name */
SQLSetDescField(ipd, 1, SQL_DESC_NAME, "@searchString",
                SQL_NTS);
SQLSetDescField(ipd, 2, SQL_DESC_NAME, "@useInflectional",
                SQL_NTS);
SQLSetDescField(ipd, 3, SQL_DESC_NAME, "@useThesaurus",
                SQL_NTS);
SQLSetDescField(ipd, 4, SQL_DESC_NAME, "@language", SQL_NTS);

/* Execute the prepared statement */
SQLExecute(stmt);
```

```
/* Retrieve the parameter names */
SQLGetDescField(ipd, 1, SQL_DESC_NAME, param_name_1,
                sizeof(param_name_1), NULL);
SQLGetDescField(ipd, 2, SQL_DESC_NAME, param_name_2,
                sizeof(param_name_2), NULL);
SQLGetDescField(ipd, 3, SQL_DESC_NAME, param_name_3,
                sizeof(param_name_3), NULL);
SQLGetDescField(ipd, 4, SQL_DESC_NAME, param_name_4,
                sizeof(param_name_4), NULL);

printf( "Procedure Input\n");
printf( "===============\n");
printf( "%s value:\t%s\n", param_name_1, search_string);
printf( "%s value:\t%d\n", param_name_2, use_inflectional);
printf( "%s value:\t%d\n", param_name_3, use_thesaurus);
printf( "%s value:\t%d\n", param_name_4, language);

printf( "\nProcedure Output\n");
printf( "===============\n");

/* Retrieve the column names for the procedure result set */
SQLColAttribute (stmt, 1, SQL_DESC_NAME, col_1_name,
                 sizeof(col_1_name), &col_1_name_length, 0);
SQLColAttribute (stmt, 2, SQL_DESC_NAME, col_2_name,
                 sizeof(col_2_name), &col_2_name_length, 0);
```

```
    printf( "%s\t\t%s\n", col_1_name,col_2_name);

    /* Fetch the procedure result set. */
    while (SQL_SUCCEEDED(ret = SQLFetch(stmt))) {
      printf( "%d\t\t\t%d\n", col_1, col_2);
    }

    /* Free up allocated handles and disconnect from the driver */
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    SQLDisconnect(dbc);
    SQLFreeHandle(SQL_HANDLE_DBC, dbc);
    SQLFreeHandle(SQL_HANDLE_ENV, env);

}
```

When run, the sample produces output similar to that shown in the following shell session:

```
$ cc -I/usr/local/easysoft/unixODBC/include/
uspSearchCandidateResumes.c -o uspSearchCandidateResumes -
L/usr/local/easysoft/unixODBC/lib/ -lodbc
$ chmod +x ./uspSearchCandidateResumes
$ ./uspSearchCandidateResumes
```

```
Procedure Input
===============
@searchString value:    ISO9000
@useInflectional value: 0
@useThesaurus value:    0
@language value:        0

Procedure Output
===============
JobCandidateID          RANK
1                       9
7                       9
10                      12
```

SQL Server Authentication Modes

Users are granted access to SQL Server instances through a SQL Server login. SQL Server provides two ways to authenticate SQL Server logins: Windows Authentication (also known as trusted connections) and SQL Server Authentication.

Windows Authentication allows users to connect to SQL Server by using their Windows user account. SQL Server uses the Windows security system to validate these trusted connections.

SQL Server authentication uses passwords stored in SQL Server to validate the connection.

Windows Authentication is Microsoft's recommended SQL Server authentication mode because it provides the following advantages:

- User names and passwords are encrypted.
- Security is easier to manage (a single Windows security model instead of a separate SQL Server security model).
- Login security improves through password expiration, minimum password lengths, and account lockout policies.

The Easysoft ODBC-SQL Server Driver supports both Windows Authentication and SQL Server Authentication. You specify the SQL Server login name and password with the `User` and `Password` data source attributes or the `UID` and `PWD` connection string attributes.

WINDOWS AUTHENTICATION

The Easysoft ODBC-SQL Server Driver asks SQL Server to use Windows Authentication to validate the connection if:

- The `User` attribute value contains a backslash (used to separate a user name from a domain), for example, `mydomain\myuser`.

– OR –

- The `Trusted_Connection` attribute is ON (set to `Yes`).

The Easysoft ODBC-SQL Server Driver passes the domain name, the user name and password to SQL Server in an encrypted form. This process involves both the Data Encryption Standard (DES) encryption method and the MD4 hashing algorithm. The Easysoft ODBC-SQL Server Driver uses open source code for both these methods. The code is distributed under the terms of the GNU Lesser General Public License (LGPL). To read the license, see `/usr/local/easysoft/sqlserver/crypt/COPYING`.

To comply with the terms of the LGPL, the encryption functions are not included in the main Easysoft ODBC-SQL Server Driver library. Instead, they are provided in the shared library file `/usr/local/easysoft/lib/libestdscrypt.so`. The Easysoft ODBC-SQL Server Driver distribution includes the source files for this library. The source files are installed in `/usr/local/easysoft/sqlserver/crypt`. The supplied Makefile will build the library on your Easysoft ODBC-SQL Server Driver platform.

SQL Server Authentication

If the `User` attribute value does not contain a backslash, the Easysoft ODBC-SQL Server Driver asks SQL Server to use SQL Server Authentication to validate the connection. The Easysoft ODBC-SQL Server Driver sends the password to SQL Server in an encrypted form, although the encryption is less strong then that used for trusted connections. The SQL Server user name is sent in plain text.

Encrypting Connections to SQL Server

SQL Server 2000 and latercan use Secure Sockets Layer (SSL) to encrypt data transmitted across a network between an instance of SQL Server and a client application.

The Easysoft ODBC-SQL Server Driver with SSL Support lets Linux and Unix applications access SQL Server 2000 and later over an encrypted connection. The SSL version of driver is included in the Easysoft ODBC-SQL Server Driver distribution and should be used instead of the standard Easysoft SQL Server driver whenever an SSL connection is required.

In this section

Accessing SQL Server over an Encrypted Connection

Read this topic if you need to connect to a SQL Server instance over an encrypted connection. Database administrators should refer first to **"Configuring and Testing SSL Encryption" on page 244** for information about setting up SSL encryption on the client and SQL Server machine.

Before following the steps in this topic, contact your database administrator for the following information:

- Is encryption enabled in the SQL Server instance (either the `Force protocol encryption` option or `ForceEncryption` option enabled)?
- Is the SQL Server instance using a self-generated SSL certificate? (Only applicable to SQL Server 2005 or later.)
- If neither of the preceding points are true, where on the Easysoft ODBC-SQL Server Driver with SSL Support machine is the root certificate authority (CA) certificate installed?

To access SQL Server over an encrypted connection

1. In `/etc/odbc.ini`, find the `SQLSERVER_SAMPLE_SSL` data source.
2. Edit the data source to connect to your SQL Server instance. For example:

```
[SQLSERVER_SAMPLE_SSL]
Driver                  = Easysoft ODBC-SQL Server SSL
Description             = Easysoft SQL Server ODBC driver
Server                  = MYSQLSERVERMACHINE\MYINSTANCE
Port                    =
Database                =
User                    = MYDOMAIN\myuser
Password                = mypassword
.
.
.
```

For more information about configuring ODBC data sources, see **"Setting Up Data Sources on Unix" on page 69**.

3. If SSL encryption is enabled on the SQL Server instance, set the `Encrypt` attribute to No, and then skip to step 5. Otherwise, skip this step.

4. Do one of the following:

   - If the SQL Server instance is using a self-generated SSL certificate, set the `TrustServerCertificate` attribute to Yes.
   - If the SQL Server instance is **not** using a self-generated certificate, edit the `CertificateFile` attribute value. Specify the file that contains the public key certificate of the root CA that signed the SQL Server SSL certificate. For example,

     `CertificateFile = /usr/share/ssl/certs/ca-bundle.crt`.

5. Use `isql` to test the data source. For example:

   cd `/usr/local/easysoft/unixODBC/bin`.

   ```
   ./isql -v SQLSERVER_SAMPLE_SSL
   ```

CONFIGURING AND TESTING SSL ENCRYPTION

Refer to this section if you are a database administrator who needs to configure and test the Easysoft ODBC-SQL Server Driver with SSL Support. The section shows you how to configure the driver to request an encrypted connection and verify that data is encrypted.

The section also contains information about setting up SSL encryption on the SQL Server machine. This information is intended to supplement rather than replace the Microsoft SQL Server documentation.

Installing an SSL Certificate

Before you can access SQL Server 2000 over an encrypted connection, an SSL certificate needs to be installed on the SQL Server machine. SQL Server 2005 or later can use an SSL certificate from a trusted CA if available or generate a self-signed certificate.

Note	Even though SQL Server 2005 or later can make encryption available without an installed SSL certificate, Microsoft recommend using a certificate signed by a trusted authority whenever possible. SSL connections that are encrypted with a self-signed certificate protect against packet sniffing but do not protect against man-in-the-middle attacks. In a man-in the-middle attack, attackers route packets through their servers, which sniff the contents as they pass through.

If an SSL certificate has not yet been installed on the SQL Server machine, obtain a certificate from a certificate vendor that meets the following requirements:

- SSL certificate requirements for SQL Server 2000:

  http://support.microsoft.com/kb/318605

- SSL certificate requirements for SQL Server 2005:

  http://blogs.msdn.com/sql_protocols/archive/2005/12/30/508311.aspx

Refer to the following Microsoft documentation for installation information:

- Installing an SSL certificate on an SQL Server 2000 machine:

  http://support.microsoft.com/kb/316898

- Installing an SSL certificate on an SQL Server 2005 or later machine:

  http://msdn.microsoft.com/en-us/library/ms189067.aspx

Testing That SSL is Available on the SQL Server Machine

The following steps show how to check that SQL Server can successfully load the SSL certificate or generate its own certificate.

To check that SSL encryption is available on the SQL Server

1. Do one of the following:
   - For SQL Server 2000, in the SQL Server Network Utility, make sure that the `Force protocol encryption` option is checked.
   - For SQL Server 2005, in SQL Server Configuration Manager, double-click SQL Server 2005 Network to expand the Protocols list. Right-click Protocols for the instance that you want to connect to and click Properties. Make sure that `ForceEncryption` is set to `Yes`.
   - For SQL Server 2008, in SQL Server Configuration Manager, double-click SQL Server Network Configuration to expand the Protocols list. Right-click Protocols for the instance that you want to connect to and click Properties. Make sure that `ForceEncryption` is set to `Yes`.
2. Restart the instance.

3. Check the SQL Server error log (*drive*:\Program Files\Microsoft SQL Server\MSSQL.*n*\MSSQL\LOG\ERRORLOG, by default) to verify that SQL Server did not report any errors when it started.

> You can verify that SQL Server has successfully generated a self-signed SSL certificate by checking the SQL Server error log for a line containing:
>
> ```
> A self-generated certificate was
> successfully loaded for encryption.
> ```
>
> **Note** If SQL Server is unable to generate a self-signed certificate, you will be unable to connect to the instance over an encrypted connection. Note that when testing SSL with SQL Server Express, we had to change the account used by the SQL Server instance from `Network Service` to `Local System` before the instance could generate a certificate.

Installing the Root CA Certificate

If encryption is enabled on the SQL Server machine, the client machine does not have to trust the CA that signed the SSL certificate used by the instance, and you can skip this section.

If you request encryption from the client machine **rather than** the SQL Server machine, the Easysoft ODBC-SQL Server Driver with SSL Support must be able to verify the ownership of the certificate used by the SQL Server instance. (Unless the SQL Server 2005 or later instance is using a self-generated certificate, in which case you should use `TrustServerCertificate` attribute to bypass the verification process.) If the server certificate was signed by a public or private CA for which the client machine does not have the public key certificate, you must install the public key certificate of the CA that signed the server certificate. If the client machine already has the public key certificate of the CA that signed the server certificate, this step is not necessary.

To install the root CA certificate on the client machine

1. On the SQL Server machine, in the Windows Run dialog box, type:

   ```
   mmc
   ```

2. In Microsoft Management Console, on the File Menu, click Add/Remove Snap-in, and then click Add.
3. In the Add Standalone Snap-In dialog box, double-click Certificates. Click Computer account when prompted and then click Next. Click Finish.
4. Click Close and then OK to close the Add/Remove Snap-in dialog boxes.

5. In the Certificates Snap-in, locate the certificate for the CA that signed the SQL Server certificate. For example, if the CA certificate is in the Trusted Root Certificate Authorities store, double-click Trusted Root Certificate Authorities and click Certificates. In the right pane of the console window, right-click the CA certificate, point to All Tasks, and then click Export.
6. Complete the Certificate Export wizard.

   When prompted to choose the export format, make sure that you choose **Base-64 encoded X.509**.
7. Use FTP to copy the exported CA certificate to the client machine from which you want to access SQL Server.

Configuring the Client to Request an Encrypted Connection

SSL encryption can be enabled either on the SQL Server machine or the client machine. If you do not want to enable encryption globally on the SQL Server machine, you can enable encryption on a per-client basis.

To configure the client to request an encrypted connection

1. Do one of the following:
   - For SQL Server 2000, in the SQL Server Network Utility, make sure that the `Force protocol encryption` option is clear.
   - For SQL Server 2005, in SQL Server Configuration Manager, double-click SQL Server 2005 Network to expand the Protocols list. Right-click Protocols for the instance that you want to connect to and click Properties. Make sure that `ForceEncryption` is set to `No`.
   - For SQL Server 2008, in SQL Server Configuration Manager, double-click SQL Server Network Configuration to expand the Protocols list. Right-click Protocols for the instance that you want to connect to and click Properties. Make sure that `ForceEncryption` is set to `No`.
2. On the machine where the Easysoft ODBC-SQL Server Driver with SSL Support is installed, create an ODBC data source to connect to the SQL Server instance.

   In your data source, the `Driver` attribute must be set to `Easysoft ODBC-SQL Server SSL`.

   Set the `Encrypt` and `TrustServerCertificate` attributes to `No`. For example:

```
[SQLSERVER_SSL_CONNECTION]
Driver      = Easysoft ODBC-SQL Server SSL
Server      = MYSQLSERVERMACHINE\MYINSTANCE
User        = MYDOMAIN\myuser
Password    = mypassword
Encrypt     = No
TrustServerCertificate= No
```

3. Use an application such as Microsoft Network Monitor, Snort or Ethereal to capture network traffic between this machine and the SQL Server machine.

   Using Network Monitor or a network sniffer tool lets you verify that you have successfully made an encrypted connection to the SQL Server machine. When testing the Easysoft driver, we used Network Monitor on the SQL Server machine and Snort on the client machine to capture network traffic.

   For information about using Network Monitor, see the Microsoft Knowledge Base article How to capture network traffic with Network Monitor **(http://support.microsoft.com/kb/148942/EN-US/**).

4. Use `isql` to connect to the data source and retrieve some data. For example:

```
$ cd /usr/local/easysoft/unixodbc/bin
$ ./isql -v SQLSERVER_SSL_CONNECTION
SQL> select * from HumanResources.EmployeePayHistory where EmployeeID = 1
```

5. In your network sniffer, verify that the data returned by the Easysoft ODBC-SQL Server Driver with SSL Support is not encrypted.

   This fragment of example output shows unencrypted data captured on the client machine by running `snort -vde`:

```
8.E.m.p.l.o.y.e
e.I.D.......=.R
a.t.e.C.h.a.n.g
e.D.a.t.e......
<.R.a.t.e......
0.P.a.y.F.r.e.q
u.e.n.c.y......
=.M.o.d.i.f.i.e
d.D.a.t.e......
```

6. Press RETURN to exit `isql`.
7. In your data source, set the `Encrypt` attribute to `Yes`.

8. Do one of the following:

   - To connect to a SQL Server instance on a machine where an SSL certificate has been provisioned, use the `CertificateFile` attribute to specify the path to the CA certificate file. The certificate file must contain the public key certificate of the CA that signed the SQL Server certificate. The public key certificate must be in base-64 PEM format.

     If the CA's public key was already installed on this machine, specify the path to the CA store that contains the public key. For example, `CertificateFile = /usr/share/ssl/certs/ca-bundle.crt`. If you exported the CA certificate on the SQL Server machine and copied it to this machine, specify the path to the certificate file. For example, `CertificateFile = /usr/share/ssl/CA/MyCA.cer`. For more information, see **"Installing the Root CA Certificate" on page 247**.

   - To connect to a SQL Server 2005 or later instance that is using a self-signed certificate, set the `TrustServerCertificate` attributes to `Yes`.

9. Use `isql` to connect to the data source and retrieve the same data as you did in step 4.

10. In your network sniffer, verify that the data returned by the Easysoft ODBC-SQL Server Driver with SSL Support is now encrypted.

    This example output shows encrypted SQL Server data captured in Snort.

```
.E..{i.2.8.G.q..
.n..{X.... 4..O.
&....Lt..Z.wrH.8
.W..{..........,
....1s_..).\k.6.
..4U..4..D...5.U
&...I......+..w.
l.W...&}x.......
....%......7...J
..C$...,j..52.~.
.w. Q.qE.Q....]4
.\.Y?...|R.VOr.S
.....K.W.. 2.#.T
.G..+..F.....T..
@"..+-.......
```

ENCRYPTING THE LOGIN PACKET

When connecting to SQL Server, the Easysoft driver passes a user name and password to the SQL Server instance. For Windows user names, a domain name is also sent. These authentication details are stored inside a login packet that is transmitted between the Easysoft driver and SQL Server.

SQL Server 2005 or later will use SSL to encrypt the login packet, if you connect with the Easysoft ODBC-SQL Server Driver with SSL Support. Unless either the client or the server instance requests encryption, the connection is not encrypted beyond the login packet.

Note	For Windows logins, SQL Server transmits the domain and user name in plain text in the response to the login packet. If you do not want this information sent in plain text, you need to enable encryption either on the SQL Server or Easysoft ODBC-SQL Server Driver with SSL Support machine.

EASYSOFT ODBC-SQL SERVER DRIVER WITH SSL SUPPORT ATTRIBUTE FIELDS

The following attributes may be set in the `odbc.ini` file:

Attribute	Description
Encrypt = Yes | No	Whether the client requests an encrypted connection to SQL Server. If you do not want to enable SSL encryption globally on the server, you can enable SSL encryption on a per client basis. To do this, set `Encrypt` to `Yes`. Do not enable SSL encryption on both the server and client, use one or the other.

Attribute	Description
TrustServerCertificate = Yes | No	Enables the client to request encryption even when an SSL certificate has not been installed on the SQL Server 2005 or later machine. If SQL Server cannot load a valid SSL certificate at startup time, it will generate a self-signed certificate to make encryption available. When the client requests encryption by setting `Encrypt` to Yes, the Easysoft ODBC-SQL Server Driver with SSL Support tries to validate the server certificate to verify the identity of the server machine. This is impossible to do with a self-signed certificate since it has not been signed by a trusted root authority. Setting `TrustServerCertificate` to Yes overrides the server validation. If the SQL Server `ForceEncryption` option is enabled, the `TrustServerCertificate` value is ignored. When encryption is enabled on the SQL Server machine, the Easysoft ODBC-SQL Server Driver with SSL Support bypasses the validation of the server certificate. Note that SQL Server 2000 cannot generate a self-signed certificate. SSL encryption is only available if the SQL Server 2000 instance is running on a computer that has a certificate assigned from a public certification authority. By default, `TrustServerCertificate` is ON (set to `Yes`).

Attribute	Description

CertificateFile = *filename*	The file that contains the public key certificate of the CA that signed the SQL Server certificate. The CA certificate file must be in base-64 PEM format.
	If the CA certificate is not installed on your client machine, you need to export the certificate on the SQL Server machine and install it on the client. For more information, see **"Installing the Root CA Certificate" on page 247**.
	Examples
	To load a CA certificate from the root CA certificate store supplied with the OpenSSL distribution, use:
	`CertificateFile = /usr/share/ssl/certs/ca-bundle.crt`
	To load a private CA certificate named `MyCA.cer` that you copied to `/usr/share/ssl/CA`, use:
	`CertificateFile = /usr/share/ssl/CA/MyCA.cer`

Attribute	Description

Cypher = *value*	The cypher suite that the Easysoft ODBC-SQL Server Driver with SSL Support will request during the SSL handshake with the SQL Server machine.
	A cypher suite is a set of authentication, encryption, and data integrity algorithms used to protect data exchanged between machines. During the SSL handshake part of the connection process, the SSL layer in the ODBC driver and the Schannel layer on the SQL Server machine negotiate to decide which cipher suite they will use.
	To see which cypher suite is being used for a particular connection, enable Easysoft ODBC-SQL Server Driver with SSL Support logging. To do this, include these lines in your ODBC data source:
	`LOGFILE = /tmp/sql-server-driver.log`
	`LOGGING = Yes`
	Connect and then examine the driver log file. Look for a log file entry similar to:
	`SSL using cypher 'RC4-MD5 SSLv3 Kx=RSA Au=RSA Enc=RC4(128) Mac=MD5'`
	This entry shows that the ODBC driver and the SQL Server machine negotiated the following cryptographic protection for the connection:
	Encryption: RC4
	Encryption strength: 128-bit
	Cryptographic checksum: MD5
	Authentication: RSA

Attribute	Description

	(You can also display the cryptographic settings negotiated during the SSL handshake by enabling Schannel logging. Enable the "`Log informational and success events`" Schannel logging option to write this information to the Windows Event Viewer logs. For information about how to do this, see **http://support.microsoft.com/kb/260729**.)
	Use the `Cypher` setting, if you want to request a different encryption or data integrity algorithm to the ones negotiated during the SSL handshake. For example:
	`# Request Triple DES (3DES) for data encryption.`
	`# Request Secure Hash Algorithm (SHA) for data`
	`# integrity protection.`
	`Cypher = 3DES+SHA`
	– OR –
	`# Request Advanced Encryption Standard (AES) for data`
	`# encryption. If AES is not available on the server,`
	`# request 3DES.`
	`Cypher = AES:3DES`
	– OR –
	`# Use Secure Hash Algorithm (SHA) to protect data`
	`# integrity. Let the SSL layers negotiate which`
	`# encryption algorithm to use.`
	`Cypher = SHA`
	If you specify a cypher suite that is not available on the server machine, the Easysoft ODBC-SQL Server Driver with SSL Support returns the error "`Required SSL (failed to receive packet)`".
	If you specify a cypher suite that the Easysoft ODBC-SQL Server Driver with SSL Support does not recognise, the driver returns the error `"SSL3_CLIENT_HELLO:no ciphers available"`.

Attribute	Description
	(For a complete list of valid `Cypher` values, see **http://www.openssl.org/docs/apps/ciphers.html**.) Note that if you are connecting to a SQL Server instance that is running in FIPS 140-2 compliance mode, the remote Schannel layer will insist that the driver uses the appropriate cypher suite. There is no need to use the `Cypher` setting in this situation. Federal Information Processing Standard (FIPS) is a U.S. government standard that defines security requirements for cryptographic modules. For more information about SQL Server and FIPS, see **http://support.microsoft.com/kb/920995**.

Attribute	Description
Entropy = *filename*	The Easysoft ODBC-SQL Server Driver with SSL Support needs a source of unpredictable data to work correctly. Many open source operating systems provide a "randomness device" (`/dev/urandom` or `/dev/random`) that serves this purpose. The Easysoft ODBC-SQL Server Driver with SSL Support tries to use `/dev/urandom` by default and will also try to use `/dev/random` if `/dev/urandom` is not available. If the driver is unable to find a suitable randomness device, it will return the error "`SSL connection failed in syscall (errno=2, there may be no source of entropy on this system, consult OpenSSL documentation)`". On systems without `/dev/urandom` or `/dev/random`, the EGD entropy gathering daemon can be used as an alternative source of random data. It provides a socket interface through which entropy (randomness) can be gathered. Use the `Entropy` attribute to specify the path to the EGD socket. For example, if you create the socket in `/etc` when you start the EGD daemon (`egd.pl /etc/entropy`), use : `Entropy = /etc/entropy`

Figure 13: Easysoft ODBC-SQL Server Driver with SSL Support data source settings.

Database Mirroring

Database mirroring is a feature introduced in SQL Server 2005 that increases data availability by creating a standby copy of a database. In database mirroring, all updates to a database (the principal database) are automatically copied to a standby database (the mirror database). If the principal database server fails, the mirror database server takes over the role of principal server and brings its copy of the database online as the principal database.

For example, Partner_A and Partner_B are two partner servers, with the principal database initially on Partner_A as principal server, and the mirror database residing on Partner_B as the mirror server. If Partner_A goes offline, the database on Partner_B can fail over to become the current principal database. When Partner_A rejoins the mirroring session, it becomes the mirror server and its database becomes the mirror database.

In this section

Making the Initial Connection to a Database Mirroring Session

To establish the initial connection to a mirrored database, a data source needs to supply the current principal server instance (known as the *initial partner*). Optionally, the data source can also supply the current mirror server instance (known as the *failover partner*). This setting is used to connect to the mirror server if the initial connection to the principal server fails. The data source must also supply the database name. The Easysoft ODBC-SQL Server Driver will not attempt to failover to the partner database if this is not done.

In the following example data source, the principal server instance for the AdventureWorks database is `123.34.45.56:4724`. The database is mirrored on `123.34.45.57:4724`.

```
[SQL Server 2005 Database Mirroring]

Driver    = Easysoft ODBC-SQL Server

# The current principal server instance.

Server    = 123.34.45.56:4724

# The current mirror server instance. If the initial attempt to
# connect the principal server fails, try to connect to this server.

Failover_Partner    = 123.34.45.57:4724

# You must specify the database to be mirrored.

Database  = AdventureWorks

# This login must have permission to access the database on both
# the principal and mirror database server.

User      = my_domain\my_username

Password  = my_password
```

For more information about setting up a data source for a mirrored database, see **"Data Source Attributes for a Mirrored Database" on page 266**.

When attempting to connect, the Easysoft ODBC-SQL Server Driver begins by using the initial partner name. If the specified server instance is available and is the current principal server instance, the connection attempt usually succeeds.

If the connection attempt to the initial partner fails, the Easysoft ODBC-SQL Server Driver tries the failover partner name, if specified. If the failover partner name is not specified, the original connection attempt continues until the network connection times out or an error is returned (just as for a non-mirrored database).

If the `Failover_Partner` attribute correctly identifies the current principal server, the Easysoft ODBC-SQL Server Driver normally succeeds in opening the initial connection.

> **Note** A database mirroring session does not protect against server-access problems that are specific to client machines, such as when a client machine is having a problems communicating with the network. A connection attempt to a mirrored database can also fail for a variety of reasons that are unrelated to the Easysoft ODBC-SQL Server Driver; for example, a connection attempt can fail because of a network error.

DATA SOURCE ATTRIBUTES FOR A MIRRORED DATABASE

This section discusses the ODBC data source attributes that are relevant for connecting to a mirrored database. For information about all Easysoft ODBC-SQL Server Driver data source attributes, see **"Attribute Fields" on page 86**.

Attribute	Notes

Server	Use the `Server` attribute to specify the current principal database server. You can use the following format for the attribute value: *`machinename`*`[\`*`instancename`*`]`. For example:
	`Server=partner_A_host`
	– OR –
	`Server=partner_A_host\instance_2`
	Note that when you specify a machine name, a DNS lookup is necessary to obtain the IP address of the server. In addition, if you specify an instance that is not listening on the default TCP port (1433), a SQL Server Browser query is also required. These lookups and queries can be bypassed by specifying the IP address and port number of the initial partner. This is recommended to minimise the possibility of external delays while connecting to that partner. To specify the IP address and port, the `Server` attribute takes the following form: `Server=`*`ip_address`*`:`*`port`* (IPv4 format) or `Server=`*`ip_address`*`*`*`port`* (IPv6 format). If you specify an IPv6 address, you also need to set the `IPv6` attribute to 1. For example:
	`# IPv4 address`
	`Server=123.34.45.56:4724`
	– OR –
	`# IPv6 address (SQL Server 2005 or later)`
	`Server=2001:4898:23:1002:20f:1fff:feff:b3a3*7022`
	`IPv6 = 1`

Attribute	Notes
Database	The data source must specify the name of the mirrored database. To do this, use the `Database` attribute. This is necessary to enable failover attempts by the Easysoft ODBC-SQL Server Driver.
Failover_Partner	Use the `Failover_Partner` setting to specify the current mirror database server. If the **initial** connection to the principal database server fails, the Easysoft ODBC-SQL Server Driver will attempt a connection to the failover partner specified by `Failover_Partner`. If the specified server is not acting as a failover partner, the connection is refused by the server. If you omit the `Failover_Partner` setting and the initial partner specified by `Server` is unavailable, the connection attempt will fail. The Failover_Partner attribute value takes the form *`machinename`*`[\`*`instancename`*`]`. For example: `Server=partner_B_host\instance_2` Alternatively, the IP address and port number of the failover partner can be supplied. If the connection attempt to the initial partner fails, the attempt to connect to the failover partner will be then be freed from relying on DNS and SQL Server Browser queries. To find out the current failover partner for a mirrored database, use `tdshelper`. For information about how to do this, see **"The Impact of a Stale Failover Partner Name" on page 270**.
User	The login that you specify with the `User` attribute must have permission to access the database on the principal **and** mirror database server. Otherwise, you will be unable to access the database if the principal role switches and the former mirror server offers its database as the principal database.

Figure 14: Data source attributes for a mirrored database

CONNECTION RETRY ALGORITHM

When you specify a failover partner with the `Failover_Partner` attribute, connection attempts are regulated by a connection retry algorithm that is specific to database mirroring. The connection retry algorithm determines the maximum time (the retry time) allotted for opening a connection in a given connection attempt.

If a connection attempt fails or the retry time expires before it succeeds, the Easysoft ODBC-SQL Server Driver tries the other partner. If a connection is not opened by this point, the Easysoft driver alternately tries the initial and failover partner names, until a connection is opened or the login period times out. The default SQL Server login timeout period is 15 seconds.

The retry time is a percentage of the login period. The retry time for a connection attempt is larger in each successive round. In the first round, the retry time for each of the two attempts is 8 percent of the total login period. In each successive round, the retry algorithm increases the maximum retry time by the same amount. For example, the retry times for the first six connection attempts is as follows:

```
8%, 8%, 16%, 16%, 24%, 24%
```

The retry time is calculated by using the following formula:

```
RetryTime = PreviousRetryTime + ( 0.08 * LoginTimeout )
```

Where *`PreviousRetryTime`* is initially 0. For example:

Round	Retry Per Attempt
1	0 + (0.08 * 15000) = 1200 msec
2	1200 + (0.08 * 15000) = 2400 msec
3	2400 + (0.08 * 15000) = 3600 msec

THE IMPACT OF A STALE FAILOVER PARTNER NAME

The database administrator can change the failover partner at any time. Therefore, a failover partner name specified in a data source might be out of date, or stale. For example, consider a failover partner named Partner_B that is replaced by another server instance, Partner_C. If the Easysoft ODBC-SQL Server Driver supplies Partner_B as the failover partner name, that name is stale. When the failover partner name is stale, the connection attempt will fail if the initial partner specified in the data source is unavailable.

To find out the current failover partner for a mirrored database, use `tdshelper`:

```
tdshelper -s initial_partner -p port -u username -a password -f
database -v
```

where:

- *`initial_partner`* is the IP address or machine name of the principal instance for the database specified with `-f`.
- *`port`* is the TCP port that the principal instance is listening on. If the instance is listening on the default port, 1433, omit `-p` *`port`*.
- *`username`* and *`password`* are the user name and password for a SQL Server login that can access the mirrored database.
- *`database`* is the mirrored database.

If the principal server instance reports the name of the failover partner, `tdshelper` displays the partner instance name in the last line of its output. For example:

```
cd /usr/local/easysoft/sqlserver/bin

./tdshelper -s my_initial_partner -u myuser -a mypassword -f my_mirroreddb -v

tdshelper: connecting to my_initial_partner

tdshelper: successfully opened connection

tdshelper: successfully logged into server with diagnostic records

tdshelper: diag record 01000:[Easysoft][ODBC SQL Server Driver][SQL
Server]Changed language setting to us_english.

tdshelper: diag record 01000:[Easysoft][ODBC SQL Server Driver][SQL
Server]Changed database context to 'my_mirroreddb'.

Connection: connected to my_initial_partner as myuser with mypassword,
database='my_mirroreddb', partner='my_failover_partner'
```

Connection Failover

Connection failover maintains data availability by allowing an application to connect to a backup SQL Server machine if the primary server is unavailable.

To configure connection failover, specify a primary server and additional fallback servers in your ODBC data source. Do this with the `Server` attribute. For example:

```
[SQL Server High Availability]
Driver          = Easysoft ODBC-SQL Server
Server          = sqlsrvhostA,sqlsrvhostB,sqlsrvhostC:1583
# This user name and password must be valid for all servers in the list.
User            = my_domain\my_user
Password        = my_password
ClientLB          = 0
```

By default, the Easysoft ODBC-SQL Server Driver will try to connect to the first server that you specify. If that server is unavailable (for example, because of a hardware or operating system failure), the Easysoft ODBC-SQL Server Driver will try to connect to next server in the list. Connection attempts continue until a connection is successfully made or until all the database servers in the list have been tried once.

To balance the load between database servers, set the `ClientLB` attribute to 1. When `ClientLB` is enabled, the server that the driver initially connects to is chosen at random.

Note that your SQL Server login (as specified by `User` and `Password`) needs to be valid on each SQL Server machine in the list. If the Easysoft ODBC-SQL Server Driver is unable to connect because SQL Server rejects the login information, the driver displays an error and does not try to connect to the next server in the list.

For more information about the `Server` and `ClientLB` attributes, see **"Attribute Fields" on page 86**.

CONNECTING TO SQL SERVER 2005 OR LATER BY USING IPV6

Internet Protocol version 6 (IPv6) is a revised version of the Internet Protocol (IP) designed primarily to address growth on the Internet. It is sometimes referred to as Internet Protocol Next Generation (IPng). The current version of IP, IP version 4 (IPv4), has proven to be robust but is over 20 years old and was not designed to support such widespread use as it does today.

The features of IPv6 include:

- 128-bit IP addresses to solve the problem of the available IP address pool being depleted.
- Extensibility to account for future growth and evolution of Internet technologies and standards.
- A simplified header format to reduce network overhead and improve performance.
- Better protection against address and port scanning attacks.
- Built-in support for Internet Protocol Security (IPsec) to prevent IPv6 traffic from being viewed or modified in transit.

IPv6 support was introduced in SQL Server 2005. Currently, SQL Server supports IPv6 on Windows XP (Service Pack 2), Windows Server 2003, Windows Vista and Windows Server 2008.

On Windows XP and 2003 Server, IPv6 is a separate component. Windows Vista and Windows Server 2008 provide an integrated IPv4 and IPv6 implementation known as the Next Generation TCP/IP stack, which is installed and enabled by default.

IPv4 and IPv6 coexist in all Windows IP implementations. Windows Vista will also run in IPv6-only mode, which means the Vista machine will only handle IPv6 traffic and is only assigned IPv6 addresses.

The Easysoft ODBC-SQL Server Driver supports both IPv4 and IPv6. It can therefore be used to connect to:

- All supported SQL Server versions by using IPv4.
- SQL Server 2005 or later by using IPv6.

CONFIGURING YOUR CLIENT MACHINE FOR IPV6

IPv6 needs to be enabled on the client machine. The procedure for this depends on the client platform. For more information, consult the IPv6 documentation for your system. For Linux systems, consult the **http://www.tldp.org/HOWTO/Linux+IPv6-HOWTO/**.

CONFIGURING YOUR ODBC DATA SOURCE FOR IPV6

To connect to a SQL Server instance that is listening on an IPv6 address, set the `IPv6` data source attribute to 1. If your DNS server or hosts file is set up to resolve IPv6 addresses, in the `Server` attribute value, specify the IPv6-enabled machine's host name. Otherwise, specify its IPv6 address. For example:

```
Server=myipv6machine\myinstance
IPv6 = 1
```

– OR –

```
Server=ABCD:EF12:0:0:0:0:0:3456\myinstance
IPv6 = 1
```

The Easysoft ODBC-SQL Server Driver supports normal and compressed IPv6 addresses. Compressed format is a short form that replaces consecutive leading zeros with two colons (::). For example, the IPv6 address shown in the previous example could be replaced with `ABCD:EF12::3456`.

The Easysoft ODBC-SQL Server Driver also supports the IPv4-mapped IPv6 address format, which is an IPv6 address that holds an embedded IPv4 address. For example, `::FFFF:192.168.19.46`.

Threading

The Easysoft ODBC-SQL Server Driver is thread safe in accordance with the ODBC specification and can safely be used behind threaded applications.

Tracing

The ODBC calls an application makes can be traced:

- Within the Driver Manager by an application.
- From within the Driver Manager.
- From within the Easysoft ODBC-SQL Server Driver.

WITHIN THE DRIVER MANAGER BY AN APPLICATION

An application can turn tracing on in the Driver Manager by using the ODBC API `SQLSetConnectAttr` (...,`SQL_ATTR_TRACE`,...).

The trace file name may also be specified with the `SQLSetConnectAttr` attribute `SQL_ATTR_TRACEFILE`.

FROM WITHIN THE DRIVER MANAGER

For the unixODBC Driver Manager, add two attributes to the [`ODBC`] section (create one if none exists) in `odbcinst.ini`.

```
Trace = Yes
TraceFile = logfile
```

For example:

```
[ODBC]
Trace = Yes
TraceFile = /tmp/unixodbc.log
```

Ensure that the user who is running the application to be traced has write permission to the log file (and to the directory containing it), or no tracing information will be produced.

FROM WITHIN THE EASYSOFT ODBC-SQL SERVER DRIVER

Driver manager trace files show all the ODBC calls applications make, their arguments and return values. Easysoft ODBC-SQL Server Driver driver tracing is specific to the Easysoft driver and is of most use when making a support call.

To enable Easysoft ODBC-SQL Server Driver logging, add a `LOGFILE` and a `LOGGING` attribute to the relevant DSN section of the `odbc.ini` file.

For example:

```
[SQLSERVER_SAMPLE]

.

.

.

LOGFILE = /tmp/sql-server-driver.log

LOGGING = Yes
```

The `LOGFILE` value is the path and file name of the log file. The value shown in the example specifies a log file named `/tmp/sql-server-driver.log`. The `LOGGING` value specifies the actions to log. The value shown in the example specifies that all actions should be logged.

Ensure that the user who is running the application to be traced has write permission to the log file (and to the directory containing it).

By default, the Easysoft ODBC-SQL Server Driver appends log information to the file specified by `LOGFILE`. If you want the driver to generate a new log file for each ODBC session, enable logging on a per process basis. To do this, change the `LOGGING` entry to:

```
LOGGING = Process
```

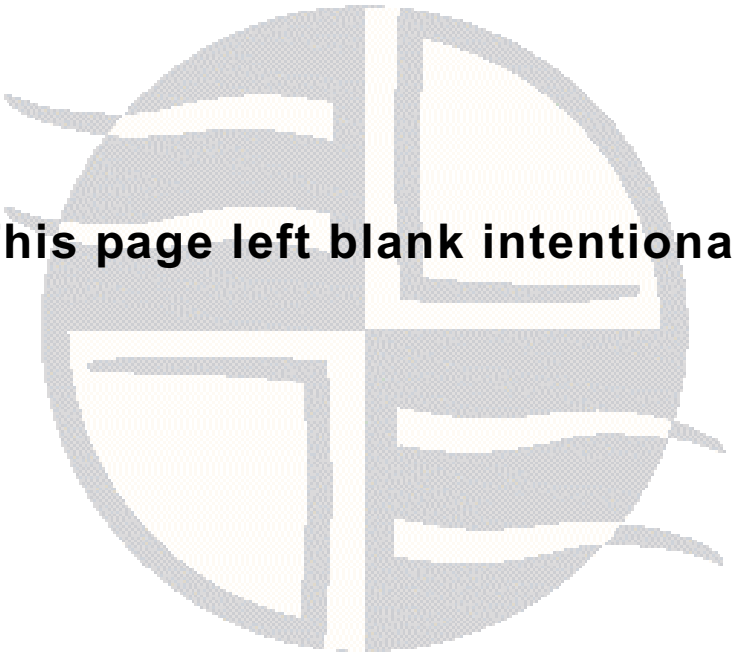
When you set `LOGGING` to `Process`, the Easysoft ODBC-SQL Server Driver creates separate log files that only contain trace output related to a particular driver process. The log file name contains the ID of the driver process (and TDS process) that the log output is associated with. For example, `/tmp/sql-server-driver.log.000076BF.B7D076B0`.

To include additional information in ODBC error messages returned by the Easysoft ODBC-SQL Server Driver, add the following line to your data source:

```
OVERRIDE = 32
```

This Driver Manager log file extract shows the extra information output by setting the `OVERRIDE` attribute. The extra information is shown in bold text.

```
DIAG [08004] [Easysoft][SQL Server Driver
10.0][server='MYMACHINE\SQLEXPRESS_ADV',proc='',class
=16,line=1,conn='mymachine\sqlexpress_adv'][Database
'AdventureWorks' does not exist. Make sure that
the name is entered correctly.
```



This page left blank intentionally

APPENDIX B GLOSSARY

Terms and definitions

Application Programmer Interface (API)

A published set of function calls and constants allowing different programmers to utilize a ready-written library of subroutines.

Authorization code

You must have an authorization code for the Easysoft product you wish to license in order to obtain a purchased license. When you purchase a product your authorization code is emailed to you. You do not need an authorization code to obtain a trial license.

Batch

A set of SQL statements submitted together and run as a group. A script is often a series of batches submitted one after the other.

Column

In an SQL table, the area in each row that stores the data value for some attribute of the object modelled by the table. For example, the Employee table in the AdventureWorks sample database models the employees of the Adventure Works Cycles company. The Title column in each row of the Employee table stores the job title of the employee represented by that row, the same way a Job Title field in a window or form would contain a job title.

Commit

An operation that saves all changes to databases, cubes, or dimensions made since the start of a transaction. A commit guarantees that all of the transaction's modifications are made a permanent part of the database, cube or dimension. A commit also frees resources, such as locks, used by the transaction.

Cursor

An entity that maps over a result set and establishes a position on a single row within the result set. After the cursor is positioned on a row, operations can be performed on that row, or on a block of rows starting at that position. The most common operation is to fetch (retrieve) the current row or block of rows.

Data Encryption Standard (DES)

A specification for encryption of computer data developed by IBM. DES uses a 56-bit key.

Data Definition Language

The subset of SQL statements that define all attributes and properties of a database and its objects. DDL statements typically start with CREATE, ALTER, or DROP.

Data Manipulation Language

The subset of SQL statements that is used to retrieve and manipulate data. DML statements typically start with SELECT, INSERT, UPDATE, or DELETE.

Data source

A database or other data repository coupled with an ODBC Driver, which has been given a Data Source Name (see **"DSN" on page 285**) to identify it to the ODBC Driver Manager.

Data type

An attribute that specifies what type of information can be stored in a column, parameter, or variable.

Database mirroring

The process of immediately reproducing every update to a read/write database (the principal database) onto a read-only mirror of that database (the mirror database) that resides on a separate instance of the Database Engine (the mirror server).

Database mirroring partners

Two server instances that act as role-switching partners for a mirrored database.

Default database

The database the user is connected to immediately after logging in to SQL Server.

Default instance

The instance of SQL Server that uses the same name as the computer name on which it is installed.

Default language

The language that SQL Server uses for errors and messages if a user does not specify a language. Each SQL Server login has a default language.

Default result set

The default mode that SQL Server uses to return a result set back to a client. Rows are sent to the client in the order in which they are placed in the result set, and the application must process the rows in this order. After running an SQL statement on a connection, the application cannot do anything on the connection except retrieve the rows in the result set until all the rows have been retrieved. The only other action that an application can perform before the end of the result set is to cancel the remainder of the result set. This is the fastest method to get rows from SQL Server to the client.

Distributed query

A single query that accesses data from multiple data sources.

Distributed transaction

A transaction that spans multiple data sources. In a distributed transaction, all data modifications in all accessed data sources are either committed or terminated.

DBMS

Database Management System -- software that handles access to a database.

Driver

See **"ODBC driver" on page 288**.

Driver Manager

Software whose main function is to load ODBC drivers. ODBC applications connect to the Driver Manager and request a data source name (DSN). The Driver Manager loads the driver specified in the DSN's configuration file. On Windows, the ODBC Data Source Administrator is used to set up the Driver Manager.

DSN

Data Source Name. A name associated with an ODBC data source. Driver Managers, such as unixODBC or the Microsoft Windows Driver Manager, use the Data Source Name to cross-reference configuration information and load the required driver.

DSN-less connection

A type of data connection that is created based on information in a data source name (DSN), but is stored as part of a project or application. DSN-less connections are especially useful for Web applications because they let you move the application from one server to another without re-creating the DSN on the new server.

Failover

In a database mirroring session, the process in which ownership of the principal role is switched from the principal server to the mirror server.

Field

A placeholder for a single datum in a record, for example you can have a Surname field in a Contact Details record. Fields are sometimes referred to as cells.

Host

A computer visible on the network.

Identity column

A column in a table that has been assigned the identity property. The identity property generates unique numbers.

Index

In a relational database, a database object that provides fast access to data in the rows of a table, based on key values. Indexes can also enforce uniqueness on the rows in a table. SQL Server supports clustered and nonclustered indexes. The primary key of a table is automatically indexed. In full-text search, a full-text index stores information about significant words and their location within a given column.

Instance

A copy of SQL Server running on a computer. A computer can run multiple instances of SQL Server 2005 or later. A computer can run only one instance of SQL Server version 7.0 or earlier, although in some cases it can also be running multiple instances of SQL Server 2000.

Integer

A data type category that includes the `bigint`, `int`, `smallint`, and `tinyint` data types.

Isolation level

The property of a transaction that controls the degree to which data is isolated for use by one process, and is guarded against interference from other processes. Setting the isolation level defines the default locking behaviour for all SELECT statements in your SQL Server session.

License key

A string that is provided by Easysoft for use in the licensing process.

Master database

The system database that records all the system-level information for an instance of SQL Server. This includes instance-wide metadata such as login accounts, endpoints, linked servers, and system configuration settings. Also, master is the database that records the existence of all other databases and the location of those database files and records the initialization information for SQL Server.

MD4

A hashing algorithm that creates a 128-bit hash value used to verify data integrity. A hashing algorithm is a mathematical procedure for randomising information to make it more secure in transmission. The more bits in a hash, the greater the security of the encryption process.

Mirror server

In a database mirroring configuration, the server instance on which the mirror database resides.

Multiple instances

Multiple copies of SQL Server running on the same computer. There can be one default instance, which can be any version of SQL Server. SQL Server 2000 or later supports multiple named instances.

Named instance

An installation of SQL Server that is given a name to differentiate it from other named instances and from the default instance on the same computer. A named instance is identified by the computer name and instance name.

NULL

An entry that has no explicitly assigned value. `NULL` is not equivalent to zero or blank. A value of `NULL` is not considered to be greater than, less than, or equivalent to any other value, including another value of `NULL`.

ODBC

Open Database Connectivity -- a programming interface that enables applications to access data in database management systems that use Structured Query Language (SQL) as a data access standard.

ODBC driver

Software that accesses a proprietary data source, providing a standardized view of the data to ODBC.

Principal server

In database mirroring, the partner whose database is currently the principal database.

Record

A group of related fields (columns) of information treated as a unit. A record is more commonly called a row in a relational database.

Result set

The set of rows returned from a SELECT statement. The format of the rows in the result set is defined by the column-list of the SELECT statement.

Row

In an SQL table, a single occurrence of the object modelled by the table. For example, in the AdventureWorks sample database, the Employee table models the employees of the Adventure Works Cycles company. Each row in the table records all the information about a specific employee such as an employee identification number, job title, and the date that employee was hired.

Server cursor

A cursor implemented on the server. The cursor itself is built at the server, and only the rows fetched by an application are sent to the client.

Server name

A name that uniquely identifies a server computer on a network. SQL Server applications can connect to a default instance of SQL Server by specifying only the server name. SQL Server applications must specify both the server name and instance name when connecting to a named instance on a server.

Structured Query Language (SQL)

A language used to insert, retrieve, modify, and delete data in a relational database, designed specifically for database queries. SQL also contains statements for defining and administering the objects in a database. SQL is the language supported by most relational databases, and is the subject of standards published by the International Standards Organization (ISO) and the American National Standards Institute (ANSI). SQL Server uses a version of the SQL language called Transact-SQL.

SQL-92

The version of the SQL standard published in 1992. The international standard is ISO/IEC 9075:1992 Database Language SQL. The American National Standards Institute (ANSI) also published a corresponding standard (Data Language SQL X3.135-1192), so SQL-92 is sometimes referred to as ANSI SQL in the United States.

Stored procedure

A precompiled collection of Transact-SQL statements that are stored under a name and processed as a unit. SQL Server supplies stored procedures for managing SQL Server and displaying information about databases and users. SQL Server-supplied stored procedures are called system stored procedures.

System databases

A set of five databases present in all instances of SQL Server that are used to store system information. The msdb database is used by SQL Server Agent to record information on jobs, alerts, and backup histories. The model database is used as a template for creating all user databases. The tempdb database stores transient objects that only exist for the length of a single statement or connection, such as worktables and temporary tables or stored procedures. The master database stores all instance-level metadata, and records the location of all other databases. The Resource database contains all the system objects that are included with SQL Server, such as system stored procedures and system tables.

Table

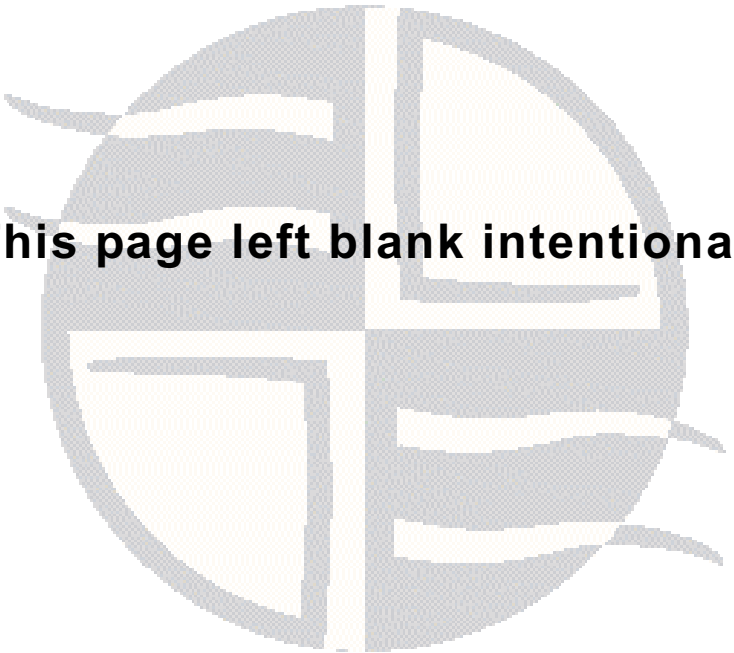
A data set in a relational database, composed of rows and columns.

Tabular Data Stream (TDS)

The SQL Server internal client/server data transfer protocol. TDS allows client and server products to communicate regardless of operating-system platform, server release, or network transport.

Transaction

A group of database operations combined into a logical unit of work that is either wholly committed or rolled back. A transaction is atomic, consistent, isolated, and durable.



This page left blank intentionally

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

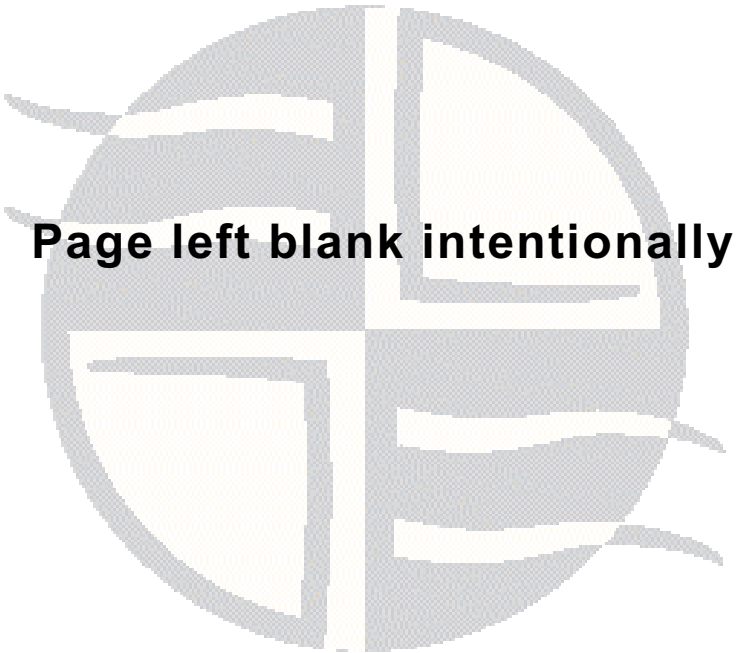
T

U

V

W

X



Page left blank intentionally