

# **Easysoft ODBC-SQL Server Driver User's Guide**

This manual documents version 2.2.n of the Easysoft ODBC-SQL Server Driver.

Copyright © 1993-2025 Easysoft Limited.

All rights reserved.

You may not reverse engineer, decompile, or disassemble this manual. Information in this document is subject to change without notice. Companies, names, and data used in examples are fictitious unless otherwise noted.

The names of companies referred to herein, their corporate logos, the names of their hardware and software may be trade names, trademarks or registered trademarks of their respective owners.

Easysoft and the Easysoft logo are registered trademarks of Easysoft Limited.

The software described in this document is provided under a [licence agreement](#) and may be used only in accordance with the terms of that agreement.

# Table of contents

Getting started .....	5
Installing the Easysoft ODBC-SQL Server Driver .....	6
Installing on Linux or UNIX .....	6
Uninstalling on Linux or UNIX .....	16
Installing on Windows .....	17
Uninstalling on Windows .....	19
Connecting to SQL Server .....	20
Connecting from Linux or UNIX .....	20
Connecting from Windows .....	22
Connection attributes .....	22
Troubleshooting database connection problems .....	46
DSN-less connections .....	49
Logging .....	50
ODBC Driver Manager logging on Linux or UNIX .....	50
Easysoft ODBC-SQL Server Driver logging on Linux and UNIX .....	50
ODBC Driver Manager logging on Windows .....	50
Easysoft ODBC-SQL Server Driver logging on Windows .....	52
Finding out what product version you have on Windows .....	53
Client applications .....	54
Microsoft Access .....	55
Linking a table .....	55
Importing a table .....	55
Microsoft Excel .....	56
Data Connection Wizard .....	56
Microsoft Query .....	56
PowerPivot .....	56
Microsoft Power BI .....	57
Oracle .....	58
Connecting SQL Server to Oracle on Windows .....	58
Connecting SQL Server to Oracle on Linux and UNIX .....	59
LibreOffice .....	62
Go .....	63
Node.js .....	64
Perl .....	66
PHP .....	69
Python .....	71
R .....	73
About the Easysoft ODBC-SQL Server Driver .....	75
ODBC API functions .....	76
Cursor support .....	79
Supported data types .....	80
The SQLGetTypeInfo function .....	80
The SQLSetConnectAttr function .....	81
The SQLSetStmtAttr function .....	84
SQL_SOPT_SS_DEFER_PREPARE .....	84
Unicode support .....	87
ANSI-Only version of the Easysoft ODBC-SQL Server Driver .....	88
The XML data type .....	89
Using large-value data types .....	91
Snapshot isolation .....	92

---

Performing bulk copy operations .....	93
bcp utility .....	93
Easysoft ODBC-SQL Server Driver extensions - bulk copy functions .....	98
Table-valued parameters .....	130
Binding procedure parameters by name .....	134
SQL Server authentication modes .....	137
Windows authentication .....	137
SQL Server authentication .....	137
Encrypting connections to SQL Server .....	138
Accessing SQL Server over an encrypted connection .....	138
Configuring and testing SSL encryption .....	139
Database mirroring .....	146
Making the initial connection to a database mirroring session .....	146
Data source attributes for a mirrored database .....	147
Connection retry algorithm .....	148
The impact of a stale failover partner name .....	149
Connection failover .....	150
Connecting to SQL Server by using IPv6 .....	151
Configuring your client computer for IPv6 .....	151
Configuring your ODBC data source for IPv6 .....	151
Finding out more about data types on Windows .....	151
Example SQL statements .....	153
Example queries .....	153
Example inserts, updates, and deletes .....	159
Index .....	162

## Getting started

This section shows you how to install the Easysoft ODBC-SQL Server Driver and configure the ODBC data source that stores the connection details for your SQL Server database. You're then ready to work with SQL Server data in your application.

- [Installing the Easysoft ODBC-SQL Server Driver](#)
- [Connecting to SQL Server](#)
- [Logging](#)

## Installing the Easysoft ODBC-SQL Server Driver

Install the Easysoft ODBC-SQL Server Driver on the computer where the application you want to connect to SQL Server is running.

- [Installing on Linux or UNIX](#)
- [Uninstalling on Linux or UNIX](#)
- [Installing on Windows](#)
- [Uninstalling on Windows](#)

### Installing on Linux or UNIX

The installation can be done by anyone with root access.

1. [Download the Easysoft ODBC-SQL Server Driver distribution for your client application platform.](#)  
If your [client application is 64-bit](#), choose the 64-bit driver distribution from the **Platforms** list. If your [client application](#) is 32-bit, choose the 32-bit driver distribution from the **Platforms** list.
2. Copy the distribution to a temporary directory on the machine where the application you want to connect to SQL Server is installed.
3. Unpack the distribution and cd into the resultant directory.
4. As root, run:

```
./install
```

5. Follow the onscreen instructions to progress through the installation.

#### Further information

- [Preinstallation requirements](#)
- [What you can install](#)
- [Where to install](#)
- [Changes made to your system](#)
- [Installing alongside other existing Easysoft product installations](#)
- [Gathering information required during the installation](#)
- [Unpacking the distribution](#)
- [License to use](#)
- [Answering questions during the installation](#)
- [Running the installer](#)
- [Locating or installing unixODBC](#)
- [Installing the Easysoft ODBC driver](#)
- [Licensing](#)
- [Testing the connection to SQL Server](#)
- [Post installation steps for non-root installations](#)
- [Setting dynamic linker search paths](#)

### Preinstallation requirements

To install the Easysoft ODBC-SQL Server Driver 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 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'll 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, refer to [Post installation steps for non-root installations](#).
- 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.

We recommend that you install all components as the root user.

## What you can install

This distribution contains:

- The Easysoft ODBC-SQL Server Driver.
- The unixODBC Driver Manager.

You 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 and Linux applications 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 ODBC Driver Manager in the standard places. If you have installed it in a non-standard location, the installation script prompts 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're 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 tries 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, refer to [Post installation steps for non-root installations](#).

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

The 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.)
2. The installation script installs an example data source into unixODBC. This data source will be added to your SYSTEM odbc.ini file. You can locate your SYSTEM odbc.ini file by using odbcinst -j.
3. Dynamic linker. On operating systems where the dynamic linker has a file listing locations for shared objects (Linux and FreeBSD), 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.
  - On FreeBSD this is usually the file /etc/defaults/rc.conf.

### Installing alongside other existing Easysoft product installations

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 are 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 get 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're 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're 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.

## Unpacking the distribution

The distribution for UNIX and Linux platforms is a tar file. To extract the installation files from the tar file, use:

```
tar -xvf odbc-sqlserver-2.2.0-linux-x86-64-unicode.tar
```

This creates 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 into the directory created by unpacking the tar file to run the installation script. For example:

```
# cd odbc-sqlserver-2.2.0-linux-x86-64-unicode
```

## License to use

The end-user license agreement (EULA) is in the file license.txt. Be sure to understand the terms of the agreement before continuing, as you're required to accept the license terms at the start of the installation.

## Answering questions during the installation

Throughout the installation, you're prompted to answer some questions. In each case, the default choice displays in square brackets and you need only press Enter to accept the default. If there are alternative responses, these are 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

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. We recommend installing as the root user. (If you're concerned about the changes that will be made to your system, refer to [Changes made to your system](#).)

To start the installation, run:

```
./install
```

You need to:

- Confirm your acceptance of the license agreement by typing "yes" or "no". For more information about the license agreement, refer to [License to use](#).
- Supply the location where the software is to be installed.

We recommend accepting the default installation path.

For more information, refer to [Where to install](#).

## Locating or installing unixODBC

We 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 applications and interfaces that support ODBC are compatible with unixODBC. The Easysoft ODBC-SQL Server Driver and any data sources that you add during the installation are 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's 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 displays:

```
Found unixODBC under path and it is version n.n.n
```

2. If the installation script can't 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 we provide.

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

Option	Description
<code>--prefix=/etc</code>	This means the default SYSTEM <code>odbc.ini</code> file where SYSTEM data sources are located is <code>/etc/odbc.ini</code> .
<code>--enable-drivers=no</code>	This means other ODBC drivers that come with unixODBC are not installed.
<code>--enable-iconv=no</code>	This means unixODBC does not look for <code>libiconv</code> . Warnings about not finding an <code>iconv</code> library were confusing our customers.
<code>--enable-stats=no</code>	Turns off 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.

Option	Description
<code>--enable-readline=no</code>	This turns off readline support in isql. We did 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 turning on readline support makes isql unusable on these systems.
<code>--prefix=/usr/local/easysoft/unixODBC</code>	This installs unixODBC into /usr/local/easysoft/unixODBC.

## Installing the Easysoft ODBC 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 displays that lists the drivers unixODBC knows about. If you're installing the Easysoft ODBC-SQL Server Driver into a different directory than it was installed before, you need to edit your `odbcinst.ini` file after the installation and correct the Driver and Setup paths. unixODBC's `odbcinst` doesn't 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, the installation script adds example data source to your `odbc.ini` file.

## Licensing

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

Licenses are stored in `installation_path/easysoft/license/licenses`.

**Important** 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 `y`, the installation runs the `licshell` script.

To obtain a license automatically, you need to be connected to the Internet and allow outgoing connections to `license.easysoft.com` on port 8884. If you're not connected to the Internet or don't allow outgoing connections on port 8884, the License Client can create a license request file that you can email to us.

When you start the License Client, the following menu displays:

```
[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're installing. The License Client then runs a program that generates a key that's 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.
- A reference number (also referred to as an authorization code). When applying for a trial license, press Enter when prompted for a reference number. This field only applies to full (paid) licenses.

You're then asked to choose 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  
The license request is output to `license_request.txt`.
- [3] Cancel this operation

If you choose to obtain the license automatically, the License Client tries to open a TCP/IP connection to `license.easysoft.com` on port 8884 and send the details you supplied along with 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 send the the request to us. The License daemon returns the license key, prints 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 sent the license request to us, we'll return a license key. Add this to the end of the file `installation_path/easysoft/license/licenses`.

## 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 don't 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 MYSQLSERVER2022HOST Port 1433 (Default)
ServerName MYSQLEXPRESSHOST\SQLEXPRESS Port 2777
ServerName MYSQLSERVER2019HOST\MYINSTANCEI Port 1510
ServerName MYSQLSERVER2019HOST\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 may not be running. `tdshelper` uses the SQL Server Browser to find out the available SQL Server instances. If the browser 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..

The example output shows that:

- The default SQL Server instance on a computer named `MYSQLSERVER2022HOST` is listening on the default SQL Server TCP port 1433.
- The default named SQL Server Express instance on a computer named `MYSQLEXPRESSHOST` is listening on port 2777.
- There are two named instances running on `MYSQLSERVER2019HOST`. 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, enter `y` to continue interactively creating your SQL Server data source.

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

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

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

Enter 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, enter a valid SQL Server user name.

Enter 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 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 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 can't 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 `drv_template` file to `odbcinst.ini`. (`drv_template` is in the directory where the Easysoft distribution was untarred to.)

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 and FreeBSD), you 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 didn't 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 creates `/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 doesn't 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 add these paths to a system file run whenever someone logs. For example, `/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 are linked against an ODBC Driver Manager, which loads the ODBC driver for your chosen data source. The dynamic linker needs to know where to find the ODBC Driver Manager shared object. The ODBC Driver Manager loads 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 attempts 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.

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.

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

```
<InstallDir>/easysoft/sqlserver:<InstallDir>/easysoft/lib
```

where <InstallDir> 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 on Linux or 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:

1. Change directory to *installation\_path*/easysoft and delete the product 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], uninstall the driver from unixODBC by using:

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

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

1. 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.
2. Remove the install.info for the Easysoft ODBC-SQL Server Driver from the /usr/local/easysoft directory.



## Installing on Windows

The Windows installation can be done by anyone with local administrator privileges.

1. [Download the Easysoft ODBC-SQL Server Driver installer.](#)
2. Follow the onscreen instructions to progress through the installation wizard.

## 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. The 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 **Restart Manager** to try to stop and restart each application or service in the list. If possible, **Restart Manager** restores applications to the same state that they were in before it shut them down.

## Licensing

By default, the installer starts the Easysoft License Manager, because you can't use the Easysoft ODBC-SQL Server Driver until you have a license. If you choose not to run Easysoft License Manager as part of the installation process, run License Manager from the **Easysoft** group in the Windows **Start** menu when you're ready to license the Easysoft ODBC-SQL Server Driver. These 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.

To license the Easysoft ODBC-SQL Server Driver:

1. In License Manager, enter your contact details.

You **must** complete the **Name**, **E-Mail Address**, and **Company** fields.

The e-mail address **must** be the same as the one used to register at the Easysoft web site. Otherwise, you won't be able to obtain a trial license.

2. Choose **Request License**.

You're prompted to choose a license type.

3. Do one of the following:

- For a trial license, choose **Time Limited Trial**, and then choose **Next**.

-Or-

- For a purchased license, choose **Non-expiring License**, and then choose **Next**.

4. Choose your product from the drop-down list when prompted, and then choose **Next**.

5. For a purchased license, enter your authorization code when prompted, and then choose **Next**.

6. Choose how to get your license when prompted.

7. Do one of the following:

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

With this method, License Manager automatically requests and then applies your license.

-Or-

- Choose **View Request**. Then open a web browser and go to [https://www.easysoft.com/support/licensing/trial\\_license.html](https://www.easysoft.com/support/licensing/trial_license.html) or [https://www.easysoft.com/support/licensing/full\\_license.html](https://www.easysoft.com/support/licensing/full_license.html), as appropriate. In the web page, enter your machine number (labelled **Number** in the license request). For purchased licenses, you also need to enter your authorization code (labelled **Ref** in the license request).

We'll automatically email your license to the email address you supplied in License Manager.

-Or-

- Choose **Email Request** to email your license request to our licensing team. Once we've processed your request, we'll email your license to the email address you supplied in License Manager.

8. Close the License Manager windows and then choose **Finish**.

If you chose either **View Request** or **Email Request**, apply your license by double-clicking the email attachment when you get the license email from us. Alternatively, start License Manager from the **Easysoft** folder in the Windows **Start** menu. Then choose **Enter License** and paste the license in the space provided.

Once you've licensed the Easysoft ODBC-SQL Server Driver, the installation is complete.

## Repairing the 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. To do this:

1. In the Windows **Taskbar**, enter Add or remove programs in the Windows **Search** box.
2. Select Easysoft ODBC-SQL Server Driver in the list, and then choose **Repair**.

## Uninstalling 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 the Easysoft ODBC-SQL Server Driver. You don't therefore need to recreate your Easysoft ODBC-SQL Server Driver data sources if you reinstall or upgrade. If you don't want to keep your Easysoft ODBC-SQL Server Driver data sources, use Microsoft **ODBC Data Source Administrator** to remove them, **before** uninstalling the Easysoft ODBC-SQL Server Driver:

1. In the Windows **Taskbar**, enter Run in the Windows **Search** box.
2. In the Windows **Run** dialog box, enter:

```
odbcad32.exe
```

3. Locate your data source in either the **User** or **System** tab.
4. Select the data source from the list, and then choose **Remove**.

If the **Remove** button isn't available, close **ODBC Data Source Administrator**, and then, in the Windows **Run** dialog box, enter:

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

Repeat the previous two steps.

### Removing the Easysoft ODBC-SQL Server Driver

1. In the Windows **Taskbar**, enter Add or remove programs in the Windows **Search** box.
2. Select Easysoft ODBC-SQL Server Driver in the list, and then choose **Uninstall**.

**Note**

Easysoft product licenses are stored 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.

## Connecting to SQL Server

Applications that support ODBC interface with an ODBC Driver Manager, which is included with the operating system, and also the Easysoft ODBC driver distribution on some platforms. One of the jobs that the ODBC Driver Manager does is to manage ODBC data sources. A data source specifies which ODBC driver to load, which data store to connect to, and how to connect to it.

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

- [Connecting from Linux or UNIX](#)
- [Connecting from Windows](#)

## Connecting from Linux or UNIX

### Creating an ODBC data source

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

- Create a SYSTEM data source, which is available to anyone who logs on to the computer where the Easysoft ODBC-SQL Server Driver is installed.
  - Or –
- Create a USER data source, which is only available to the user who is currently logged on to the computer where the Easysoft ODBC-SQL Server Driver is installed.

By default, the Easysoft ODBC-SQL Server Driver installation creates a sample SYSTEM data source named `SQLSERVER_SAMPLE`. If you're 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 SQL Server, USER data sources must be created and edited in `$HOME/.odbc.ini`.

### Notes

- 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.

The Driver attribute identifies the ODBC driver in the `odbcinst.ini` file to use for a data source. When the Easysoft ODBC-SQL Server Driver is installed into `unixODBC`, it places a Easysoft ODBC-SQL Server entry into the `odbcinst.ini` file. You should always have `Driver = Easysoft ODBC-SQL Server` in your Easysoft ODBC-SQL Server Driver data sources therefore.

The Easysoft ODBC-SQL Server Driver distribution includes two drivers:

- One with SSL support that should be used if you want to access SQL Server over an encrypted connection.
- One without SSL support that should be used for SQL Server when encryption is not required.

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 the Easysoft ODBC-SQL Server Driver with SSL Support data sources, refer to [Encrypting connections to SQL Server](#).

To configure a Easysoft ODBC-SQL Server Driver 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 Driver

# 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 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
```

The Easysoft ODBC-SQL Server Driver must be able to find the following shared objects:

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

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.

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. Enter `./isql -v data_source`, where `data_source` is the name of the target data source.
3. At the prompt, enter an SQL query. For example:

```
SQL> SELECT * FROM Suppliers;
```

–Or–

4. Enter `help` to return a list of tables:

```
SQL> help
```

## Connecting from Windows

### Creating an ODBC data source

1. In the Windows **Taskbar Search** box, enter “Run”.
2. Do one of the following:
  - If your application is 64-bit, in the **Run** dialog box, enter:

```
odbcad32.exe
```

-Or-

- If your application is 32-bit, in the **Run** dialog box, enter:

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

#### Note

If you're not sure whether your application is 32-bit or 64-bit, start your application, then in Windows **Task Manager** check whether your application's process name contains (32-bit). For example, the process name for the 32-bit version of Excel is Microsoft Excel (32-bit); the process name for the 64-bit version of Excel is Microsoft Excel. On older versions of Windows, 32-bit applications contain \*32 in the process name rather than (32-bit). For applications such as Oracle or SQL Server that run as a service, check the \*Background processes\* list rather than the **Apps** list in **Task Manager**. If you're running a programming language from within a Windows command-line shell (for example, Command or PowerShell), in your shell, run the .exe file for the programming language. For example, run perl, php, python, or node. In **Task Manager**, expand the process list for **Windows Command Processor** or **Windows PowerShell**, as appropriate, and check whether the process for your programming language contains (32-bit).

3. Do one of the following:
  - To create a data source that only the user you're currently logged in as can access, choose the **User** tab.  
If your application is a Windows service (for example, SQL Server or Oracle) creating a user data source won't work, unless the service is running as the same user you're logged in as.
  - To create a data source that all users on this computer can access, choose the **System** tab.
4. Choose **Add**.
5. In the list of ODBC drivers, select Easysoft ODBC-SQL Server Driver, and then choose **Finish**.
6. Complete the Easysoft ODBC-SQL Server Driver configuration dialog box.  
To find out how to do this, refer to the Connection attributes section.
7. To test the connection to SQL Server, choose **Test**.  
Note that this doesn't test that the Easysoft ODBC-SQL Server Driver is licensed. If you haven't yet [licensed](#) the Easysoft ODBC-SQL Server Driver, this ODBC data source won't work with your application, even if the **Test** button succeeds.

## Connection attributes

- [Setting on Linux and UNIX](#)
- [Setting on Windows](#)

## Setting on Linux and UNIX

To configure a Easysoft ODBC-SQL Server Driver 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 Driver

# 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 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
```

For more information about these mandatory attributes and other optional attributes, refer to this table:

Attribute	Description
Driver = <i>value</i>	The name of the ODBC driver to use with this data source. To connect to a SQL Server 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	Some applications display this to help users identify a particular data source.

Attribute	Description
Server = <i>value</i>	<p>The SQL Server instance that you want to connect to. To connect to the default SQL Server instance, enter:</p> <pre>computername</pre> <p>where <i>computername</i> is the name or IP address of the host where SQL Server is running.</p> <p>Note that if you're connecting to a SQL Server or later instance that's listening on an IPv6 address, set the IPv6 attribute to 1.</p> <p>To connect to a named instance, enter:</p> <pre>computername\instancename</pre> <p>where <i>instancename</i> is the SQL Server instance.</p> <p>To connect to the default SQL Server Express named instance, enter:</p> <pre>computername\sqlexpress</pre> <p>On Windows, you also have the option of using a named pipe connection. To use named pipes, use this format for the Server value:</p> <pre>\\localhost\pipe\MSSQL\$instance\sql\query</pre> <p>For example:</p> <pre>\\localhost\pipe\MSSQL\$sqlexpress\sql\query</pre> <p><b>Connection failover</b></p> <p>If your SQL Server database is available on more than one SQL Server computer, 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:</p> <pre>Server = primaryserver[:port] [, fallbackserver[:port]...]</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• <i>primaryserver</i> is the name or IP address of the primary SQL Server computer on which your database is available.</li> <li>• <i>port</i> 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.</li> <li>• <i>fallbackserver</i> is the name or IP address of an alternative SQL Server computer on which your database is available.</li> </ul> <p>For example:</p> <pre>Server = sqlsrvhostA,sqlsrvhostB,sqlsrvhostC:1583</pre>



Attribute	Description
Server = <i>value</i>	<p>Connection attempts continue until either a connection is successfully made or all the servers in the list have been tried once.</p> <p>Note that your SQL Server login (as specified by User and Password) needs to be valid on each SQL Server computer in the list. The SQL Server login must have permission to access the database on each SQL Server computer.</p> <p>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.</p>
Port = <i>num</i>	<p>The TCP port that SQL Server is listening on.</p> <p>If you're connecting to a default instance that's listening on port 1433, the Port setting can be omitted.</p> <p>If the SQL Server Browser service is running, the Easysoft ODBC-SQL Server Driver will automatically detect the port number and the Port setting can be omitted.</p> <p>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.</p> <p>If the SQL Server Browser 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.</p> <p>If your database administrator has hidden the SQL Server instance from the SQL Server Browser, you need to specify the port number of the hidden instance.</p> <p>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.</p>

Attribute	Description
User = <i>value</i>	<p>The SQL Server login name to use when connecting to SQL Server.</p> <p>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:</p> <pre data-bbox="469 360 1477 427">domain\username</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• <i>domain</i> is the name of the Windows domain that the SQL Server computer is in or one that the SQL Server computer trusts.</li> <li>• <i>username</i> is the user name of a user who belongs to this domain.</li> </ul> <p>If the SQL Server instance permits SQL Server Authentication, you can also specify a SQL Server user name.</p> <p>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, refer to <a href="#">DSN-less Connections</a>.</p> <p>On Windows, if you want to access SQL Server with a Windows log in, choose <b>With Integrated Windows Authentication</b>.</p>
Password = <i>value</i>	<p>The password for the login name specified by User.</p> <p>To specify the login password in the connection string, use PWD rather than Password.</p>
Authentication = <i>value</i>	<p>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're connecting to SQL Azure with an Azure Active Directory account user name and password.</p>

Attribute	Description
ServerName = <i>value</i>	<p>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:</p> <pre>xyz12345yzx.database.windows.net</pre> <p>There are two ways to specify the SQL Azure server, as shown by the following data source extracts:</p> <pre>ServerName = xyz12345yzx.database.windows.net User = myuser</pre> <p>-Or-</p> <pre>Server = xyz12345yzx.database.windows.net User = myuser@xyz12345yzx</pre> <p>For more information about SQL Azure, refer to the following Easysoft tutorial:  <a href="https://www.easysoft.com/products/data_access/odbc-sql-azure-driver/linux-unix.html">https://www.easysoft.com/products/data_access/odbc-sql-azure-driver/linux-unix.html</a></p>
Database = <i>value</i>	<p>The default database to use for the connection.</p> <p>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 computer 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).</p> <p>If the database does not exist or the login does not have permission to access the database, the connection will fail.</p> <p>Note that using the default database for the login ID is more efficient than specifying a default database in the ODBC data source.</p>

Attribute	Description
QuotedId = Yes   No	<p>When turned 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:</p> <pre data-bbox="477 398 1074 510">SELECT CompanyName FROM "Customer and Suppliers by City" WHERE City = 'New York'</pre> <p>If a single quotation mark is part of the literal string, it can be represented by two single quotation marks.</p> <p>When turned off (set to No), 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.</p> <p>For more information about the QUOTED_IDENTIFIERS option, refer to the SQL Server Transact-SQL documentation.</p> <p>By default, QuotedId is turned on.</p>
User_Domain = <i>value</i>	<p>If your user name contains a domain, specify the domain with this attribute. So, myuser@mydomain becomes:</p> <pre data-bbox="477 1120 831 1189">User = myuser User_Domain = mydomain</pre> <p>Omit the @ symbol.</p> <p>Some applications (for example Oracle Heterogeneous Services) don't support the use of domains in user names.</p>

Attribute	Description
OFFAnsiNPW = Yes   No	<p>When turned on (set to Yes), the ANSI_NULLS, ANSI_WARNINGS, and ANSI_PADDING options are set to ON for the connection.</p> <p>When ANSI_NULLS is turned 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:</p> <pre data-bbox="475 405 1476 544">SELECT * FROM MyTable WHERE MyColumn IS NULL</pre> <p>The Transact-SQL syntax = NULL and &lt;&gt; NULL are not supported.</p> <p>When ANSI_NULLS is turned off, the equals (=) and not equal to (&lt;&gt;) comparison operators must be used to make comparisons with NULL and non NULL values in a table.</p> <p>When ANSI_WARNINGS is turned 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.</p> <p>When ANSI_PADDING is turned on, trailing blanks on VARCHAR values and trailing zeroes on VARBINARY values are not automatically trimmed.</p> <p>For more information about the ANSI_NULLS, ANSI_WARNINGS, and ANSI_PADDING options, refer to the SQL Server Transact-SQL documentation.</p> <p>By default, AnsiNPW is turned on.</p>
Language = <i>value</i>	<p>The national language to use for SQL Server system messages. Use this format:</p> <pre data-bbox="475 1417 1476 1485">Language = <i>language</i></pre> <p>where <i>language</i> is one the language aliases contained in the sys.syslanguages table.</p> <p>For example:</p> <pre data-bbox="475 1659 1476 1727">Language = French</pre> <p>If no language is specified, the connection uses the default language specified for the login on the server.</p>

Attribute	Description
Appname = <i>value</i>	<p>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:</p> <pre data-bbox="469 309 1477 371">Appname = isql</pre> <p>The default value is ODBC.</p> <p>SQL Server stores the application name in the master.dbo.sysprocesses column program_name. The name is returned by the APP_NAME function.</p>
MARS_Connection = Yes   No	<p>When turned on (set to Yes), multiple active result sets (MARS) are enabled on the connection. 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.</p> <p>For non-MARS connections (MARS_Connection turned off), 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.</p> <p>By default, MARS_Connection is turned off.</p>
Logging = Yes   No	<p>Whether Easysoft ODBC-SQL Server Driver logging is turned on. To turn on Easysoft ODBC-SQL Server Driver logging, add a Logging=Yes entry to the relevant DSN section of the odbc.ini file.</p>
LogFile = <i>value</i>	<p>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).</p>

Attribute	Description
PreserveCursor = Yes   No	<p>When turned on (set to Yes), the Easysoft ODBC-SQL Server Driver preserves cursors when <a href="#">SQLEndTran</a> commits or rolls back a transaction.</p> <p>By default, PreserveCursor is turned off, which means that cursors are closed when a transaction is committed or rolled back by using <a href="#">SQLEndTran</a>.</p> <p>This behaviour can also be configured by setting <a href="#">SQL_COPT_SS_PRESERVE_CURSORS</a> with <a href="#">SQLSetConnectAttr</a>. For more information and a code sample, refer to <a href="#">SQL_COPT_SS_PRESERVE_CURSORS</a>.</p>
Wsid = <i>value</i>	<p>The workstation ID. The default value is the host name of the computer where the ODBC application is running. SQL Server stores the workstation ID in the master.dbo.sysprocesses column hostname. The ID is returned by <a href="#">sp_who</a> and the <a href="#">HOST_NAME</a> function.</p>
Version7 = Yes   No	<p>Set to Version7 to Yes if you're connecting to a SQL Server 7.0 database.</p> <p>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 faster connection time.</p> <p>By default, Version7 is turned off (set to No).</p>
ForceShiloh = Yes   No	<p>When turned on (set to Yes), the Easysoft ODBC-SQL Server Driver assumes that it's connecting to a SQL Server 2000 instance and only uses the SQL Server 2000 version of TDS to communicate with the instance.</p> <p>By default, ForceShiloh is off (set to No).</p>

Attribute	Description
ClientLB = Yes   No	<p>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.</p> <p>When ClientLB is turned 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.</p> <p>When ClientLB is turned 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.)</p> <p>Example:</p> <p>You specify a primary server (sqlsrvhostA) and two fallback servers (sqlsrvhostB and sqlsrvhostC):</p> <pre>Server = sqlsrvhostA,sqlsrvhostB,sqlsrvhostC:1583</pre> <p>When ClientLB is turned 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).</p> <p>When ClientLB is turned off, the Easysoft driver will try to connect to sqlsrvhostA and then sqlsrvhostB (if sqlsrvhostA is unavailable) and finally sqlsrvhostC (if sqlsrvhostB is unavailable).</p>
Failover_Partner = <i>value</i>	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.
MultiSubnetFailover = Yes   No	<p>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.</p> <p>By default, MultiSubnetFailover is turned off (set to No).</p>
ApplicationIntent = ReadOnly   ReadWrite	Specifies the application workload enter 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 = <i>num</i>	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 = <i>num</i>	Specifies the number of seconds between each connection retry attempt. Valid values are 1-60. The default value is 10 seconds,



Attribute	Description
VarMaxAsLong = Yes   No	<p>When turned 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 turned on (set to Yes), the Easysoft ODBC-SQL Server Driver returns a VARCHAR(MAX) column as a SQL_LONGVARCHAR.</p> <p>By default, VarMaxAsLong is turned off.</p>
VarMaxAsVarchar = Yes   No	<p>When turned 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.</p> <p>By default, VarMaxAsVarchar is turned off.</p>
DisguiseGuid = Yes   No	<p>When turned 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 don't recognise UNIQUEIDENTIFIER types and therefore fail to return data from tables containing these column types.</p> <p>By default, DisguiseGuid is turned off (set to No).</p>
DisguiseLong = Yes   No	<p>When turned 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.</p> <p>By default, DisguiseLong is turned off (set to No).</p>
LimitLong = <i>num</i>	<p>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.</p> <div> <p><b>Note</b> LimitLong only has an effect on MAX data, if VarMaxAsLong is turned on (set to Yes).</p> </div> <p>By default, LimitLong is turned off (set to No).</p>
DPrec = <i>num</i>	<p>The precision to use when converting FLOAT(25-53) data in a result set to a string</p> <p>If an application specifies a string as the target type for non-character data in a <a href="#">SQLBindCol</a> or <a href="#">SQLGetData</a> 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.</p> <p>The default precision is 6.</p>

Attribute	Description
FPrec = <i>num</i>	<p>The precision to use when converting FLOAT(1-24) or real data in a result set to a string</p> <p>If your application specifies a string as the target type for non-character data in a <a href="#">SQLBindCol</a> or <a href="#">SQLGetData</a> 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.</p> <p>The default precision is 6.</p>
Strftime = <i>format</i>	<p>The format to use when converting ftimestamp data in a result set to a string.</p> <p>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.</p> <p>The Easysoft ODBC-SQL Server Driver uses strftime to do the conversion, and so the format should be one of the format strings supported by strftime. For the available format strings, refer to the strftime(3) man page.</p> <p>For example, the format string specified in the following line:</p> <pre>STRFTIME = %d %h %Y %T</pre> <p>would produce:</p> <pre>11 March 2025 12:35:29</pre> <p>given this SQL statement:</p> <pre>SELECT CAST('2025-03-11 12:35:29.123' AS datetime)</pre>
Strfsize = <i>num</i>	<p>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.</p> <p>For example, to set the display size to 32, add the following line to your data source:</p> <pre>STRFSIZE = 32</pre>

Attribute	Description
ConvToUtf = Yes   No	<p>When turned 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.</p> <p>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.</p> <pre>ConvToUtf = Yes</pre> <p>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:</p> <pre>SQLPrepare( hstmt, "INSERT INTO MYNCHARTABLE VALUES (?)", SQL_NTS ); SQLBindParameter( hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_WCHAR, 100, 0, &amp;cval, sizeof( cval ), &amp;len1 );</pre> <p>By default, ConvToUtf is turned off (set to No).</p> <p><b>Example: Retrieving data</b></p> <p>We ran LibreOffice 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 Base and ran this SQL:</p> <pre>USE Northwind SELECT CompanyName FROM Suppliers WHERE SupplierID = 29</pre> <p>SQL Server stores data in the CompanyName column as a UCS-2 encoded NVARCHAR type.</p> <p>The results for this query should be:</p> <pre>Forêts d'érables</pre> <p>Instead, we got these results:</p> <pre>For?ts d'?rables</pre> <p>The ? symbols indicate that application was unable to convert the character from the server encoding scheme to the client encoding scheme.</p> <p>In LibreOffice Writer, we used the <b>Insert &gt; Special Character</b> command to insert ê and é into a new document. We did this to show that LibreOffice 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:</p>

Attribute	Description
ConvToUtf = Yes   No	<div data-bbox="477 188 1477 280"> <pre>\$ file ooo_chars.txt ooo_chars.txt: Unicode text, UTF-8</pre> </div> <p data-bbox="451 293 1497 371">The file command's output indicates that the encoding scheme LibreOffice was using is UTF-8.</p> <p data-bbox="451 385 1497 539">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 LibreOffice expects.</p> <p data-bbox="451 553 1497 600"><b>Example: inserting data</b></p> <p data-bbox="451 613 1497 692">We created a SQL file named insert-northwind-shipper.sql on a Ubuntu computer:</p> <div data-bbox="477 719 1477 940"> <pre>-- Insert new record into the Northwind shippers table USE Northwind; INSERT INTO Shippers (CompanyName, Phone) VALUES (N' '&gt; Diamond Shipping', '(11) 555-2167'); SELECT * FROM Shippers;</pre> </div> <p data-bbox="451 954 1497 1189">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 used 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:</p> <div data-bbox="477 1216 1477 1321"> <pre>\$ file insert-northwind-shipper.sql insert-northwind-shipper.sql: Unicode text, UTF-8</pre> </div> <p data-bbox="451 1335 1497 1413">In the same shell, we used insert-northwind-shipper.sql as an input file to isql:</p> <div data-bbox="477 1440 1477 1547"> <pre>/usr/local/easysoft/unixODBC/bin/isql -v SQLSERVER_SAMPLE &lt; insert-northwind-shipper.sql</pre> </div> <p data-bbox="451 1561 1497 1758">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 computer was capable of rendering the ☐ character:</p>

Attribute	Description
ConvToUtf = Yes   No	<pre> SQL&gt;--+-----+-----+   ID   CompanyName   Phone   +-----+-----+-----+   1   Speedy Express   (503) 555-9831     2   United Package   (503) 555-3199     3   Diamond Shipping   (11) 555-2167   +-----+-----+-----+ </pre>
ConvWToUtf = Yes   No	<p>When turned 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.</p> <p>By default, ConvWToUtf is turned off (set to No).</p>
SQLServerUTF = Yes   No	<p>When turned 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 <a href="#">SQLConnectW</a>, <a href="#">SQLDriverConnectW</a>, and <a href="#">SQLBrowseConnect</a>. The SQLServerUTF attribute must be specified in a section named ODBC in <code>odbc.ini</code>. For example:</p> <pre> [ODBC] SQLServerUTF = Yes </pre> <p>By default, SQLServerUTF is turned off (set to No).</p>
UTF8DB = Yes   No	<p>If you have set a UTF-8 collation at instance, database, or column level, add this entry to your data source:</p> <pre> UTF8DB = Yes </pre> <p>UTF-8 collations were introduced in SQL Server 2019 and have the suffix UTF8.</p>

Attribute	Description
Client_CSet = <i>encoding</i>	<p data-bbox="451 163 1497 241">Specifies the encoding on the Easysoft ODBC-SQL Server Driver computer.</p> <p data-bbox="451 253 1497 443">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:</p> <pre data-bbox="475 477 1473 622"># Convert from EUC-JP when submitting data to SQL Server # Convert to EUC-JP when retrieving data from SQL Server Client_CSet = EUC-JP</pre> <p data-bbox="451 633 1497 790">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.</p> <p data-bbox="451 801 1497 925">Use Client_CSet if you experience data loss or corruption when working with character data and your application cannot convert data to the encoding scheme it expects.</p> <p data-bbox="451 936 1497 1093">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 computer where the Easysoft ODBC-SQL Server Driver is installed:</p> <pre data-bbox="475 1126 1473 1193">iconv -l</pre> <p data-bbox="451 1205 1497 1361">Set Client_CSet to the encoding that corresponds with the LANG environment variable value on the client computer. For example, if LANG was set to en_US.UTF-8 on the client computer, you would set Client_CSet to UTF-8.</p> <p data-bbox="451 1373 1497 1496">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 turned on):</p> <pre data-bbox="475 1529 1473 1630">One or more characters in the input stream could not be converted</pre> <p data-bbox="451 1641 1497 1765">Note that if your client computer encoding is UTF-8, you can use Client_CSet as an alternative to ConvToUTF. You do not need to set both data source attributes.</p> <p data-bbox="451 1776 1497 1888">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 computer encoding.</p>

Attribute	Description
Server_CSet = <i>encoding</i>	<p data-bbox="456 161 1490 407">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:</p> <pre data-bbox="475 456 1276 568">INSERT INTO my_table(my_char_col, my_varchar_col, my_text_col) VALUES (?, ?, ?) SELECT * FROM my_table WHERE my_varchar_col = ?</pre> <p data-bbox="456 595 1490 860">Use Server_CSet if you experience data loss or corruption when working with data stored in CHAR, VARCHAR, and TEXT columns. The Easysoft ODBC-SQL Server Driver links to iconv on your computer 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:</p> <pre data-bbox="475 909 1331 981">SELECT COLLATIONPROPERTY('collation' , 'CodePage') AS CodePage</pre> <p data-bbox="456 1008 1490 1137">where <i>collation</i> is the SQL Server database collation, if set, otherwise the SQL Server instance collation. In the following example, collation would be Cyrillic_General_CI_AS.</p> <pre data-bbox="475 1187 1315 1653">SELECT DATABASEPROPERTYEX('MyDatabase', 'Collation') SQLCollation;  SQLCollation ----- NULL  SELECT SERVERPROPERTY('Collation') SQLCollation;  SQLCollation ----- Cyrillic_General_CI_AS</pre> <p data-bbox="456 1680 1490 1760">For a list of iconv encodings, run this command on the computer where the Easysoft ODBC-SQL Server Driver is installed:</p> <pre data-bbox="475 1809 612 1836">iconv -l</pre> <p data-bbox="456 1863 1490 2110">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.</p>

Attribute	Description
Server_UCSet = <i>encoding</i>	<p>Specifies the SQL Server encoding for character data.</p> <p>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.</p> <p>The default Unicode encoding is UTF-16le.</p>
Use_LCID = Yes   No	<p>Whether to automatically work out which character set to use based on the SQL Server column or instance Locale ID (LCID).</p> <p>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.</p> <p>The ODBC application must bind the character data as a Unicode data type (for example, a SQL_WCHAR).</p> <p>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.</p> <p>If your character set is not supported, the Client_CSet and Server_CSet attributes provide an alternative conversion mechanism.</p> <p>By default, Use_LCID is turned off (set to No).</p>
LCID = <i>localeid</i>	<p>Sets the ClientLCID the parameter in the TDS login packet. The ClientLCID parameter is described in the TDS protocol specification:</p> <p><a href="https://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-TDS%5D.pdf">https://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-TDS%5D.pdf</a></p>
Trusted_Domain = <i>value</i>	<p>The Windows domain that the user specified with User belongs to.</p> <p>If the user belongs to the same domain as the one that the SQL Server computer is in, you can omit Trusted_Domain. The Easysoft ODBC-SQL Server Driver automatically detects the domain in this case.</p> <p>If you specify a Domain with Trusted_Domain, set Trusted_Connection to Yes and omit the domain from User. For example:</p> <pre data-bbox="467 1825 1484 2049"># Windows authentication User = mylocalcomputeruser Password = mypassword Trusted_Connection = 1 Trusted_Domain = mycomputername</pre>



Attribute	Description
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 (turned off).
IPv6 = Yes   No	<p>Set IPv6 to Yes when connecting to a SQL Server instance that's listening on an IPv6 address.</p> <p>By default, IPv6 is turned 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.</p> <p>For more information about IPv6, refer to <a href="#">Connecting to SQL Server by using IPv6</a>.</p>
ConnectionTimeout = <i>num</i>	<p>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 computer has been established, the Easysoft ODBC-SQL Server Driver will wait <i>num</i> 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".</p> <p>The default value 0 means that no connection timeout is applied by the Easysoft ODBC-SQL Server Driver.</p> <p>A timeout set by calling <a href="#">SQLSetConnectAttr</a> with the SQL_ATTR_CONNECTION_TIMEOUT connection attribute will override ConnectionTimeout.</p>

Attribute	Description
LogonTimeout = <i>num</i>	<p>The number of milliseconds to wait for a TCP connection to the SQL Server computer to be established before returning to the application. When you define a timeout, the initial connection phase lasts for <i>num</i> milliseconds. If the Easysoft ODBC-SQL Server Driver is unable to connect to the target SQL Server computer 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".</p> <p>The default value 0 means that no initial connection timeout is applied by the Easysoft ODBC-SQL Server Driver.</p> <p>The Easysoft ODBC-SQL Server Driver classes the connection phase as obtaining the IP address of the SQL Server computer and connecting to it. This means that if you specify the Server attribute value as a computer 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 computer 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 computer as an IP address or tell your resolver library to consult /etc/hosts before DNS and place an entry in /etc/hosts.</p> <p>A timeout set by calling <a href="#">SQLSetConnectAttr</a> with the SQL_ATTR_LOGIN_TIMEOUT connection attribute will override LogonTimeout.</p>
RcvBuffer = <i>num</i>	<p>The size of the receive buffer for the socket in bytes. Possible values for <i>num</i> are:</p> <p>0, do not set the receive buffer size, use the system default value.</p> <p><i>n</i>, where <i>n</i> is a number greater than 0, set the receive buffer to the specified size by passing <i>n</i> to the setsockopt() function .</p> <p>By default, the system default receive buffer size is used.</p>
SoKeepalive = Yes   No	<p>Whether to use TCP keepalive probes to verify that an idle connection is still intact.</p> <p>When turned on (set to Yes), keepalive probes are sent, after a period of inactivity, to verify that the connection to the SQL Server computer 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.</p> <p>The duration of the period of inactivity is a system default, and is typically two hours.</p> <p>By default, SoKeepalive is turned off (set to No).</p>

Attribute	Description
PacketSize = <i>num</i>	<p>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.</p> <p>The default packet size is 4096 bytes.</p>
ColumnEncryption = Enabled   Disabled	<p>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.</p> <p>Always Encrypted columns were introduced in SQL Server 2016. For more information, refer to:</p> <p><a href="https://www.easysoft.com/blog/sql-server-always-encrypted.html">https://www.easysoft.com/blog/sql-server-always-encrypted.html</a></p>
Allow_C_Comment = Yes   No	<p>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 (/* ... */).</p> <p>By default, the Easysoft ODBC-SQL Server Driver strips single line /* ... */ comments from SQL statements before passing the SQL to SQL Server. For example:</p> <pre data-bbox="469 981 1477 1084">SELECT ContactID, /* FirstName, */ LastName FROM Person.Contact</pre> <p>becomes:</p> <pre data-bbox="469 1164 1477 1227">SELECT ContactID, LastName FROM Person.Contact</pre> <p>To preserve single line /* ... */ comments in the SQL that is passed to SQL Server, set the Allow_C_Comment attribute to Yes.</p>
XATimeout = <i>num</i>	Sets the timeout in seconds 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, refer to the Easysoft blog on the Easysoft web site.
XASocketTimeout = <i>num</i>	Sets the socket timeout in seconds for an XA transaction created by an xa_open call initiated by the Easysoft ODBC-SQL Server Driver. If not set, the socket timeout defaults to the XATimeout value + 5.

Attribute	Description
Kerberos = Yes   No	<p>Whether to access a SQL Server instance as a Kerberos service.</p> <p>When turned on (set to Yes), the Easysoft ODBC-SQL Server Driver will attempt to obtain a service ticket for the following Service Principal Name (SPN):</p> <pre data-bbox="469 360 1477 427">MSSQLSvc/server:port</pre> <p>where:</p> <ul style="list-style-type: none"> <li>server is the name or IP address of the SQL Server computer specified with the Server attribute.</li> <li>port is the port on which the SQL Server instance is listening, which is specified with the Port attribute.</li> </ul> <p>The ServerSPN attribute provides an alternative way to supply a service principal name.</p> <p>Don't 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 computer. For more information about kinit and accessing SQL Server as a Kerberos service, refer to the following Easysoft tutorial:</p> <p><a href="https://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html">https://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html</a></p> <p>By default, Kerberos is turned off (set to No), and the Easysoft ODBC-SQL Server Driver will use either SQL Server or Windows authentication (refer to <a href="#">SQL Server authentication modes</a>) to validate a user specified in the data source.</p>
ServerSPN = <i>value</i>	<p>The Service Principal Name (SPN) for a SQL Server instance that has been registered as a Kerberos service.</p> <p>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.</p> <p>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.</p> <p>If the SPN contains an instance name (for example, MSSQLSvc/mysqlservercomputer:myinstance), you need to use the ServerSPN attribute rather than the Kerberos attribute.</p> <p>Do not specify a User or Password value in the data source if you specify a ServerSPN value.</p> <p>On Windows, this attribute is labelled SPN.</p>

Attribute	Description
FailoverServerSPN = <i>value</i>	<p>The SPN for a mirror server instance that has been registered as a Kerberos service. For more information about Kerberos and Database Mirroring, refer to this Easysoft tutorial:</p> <p><a href="https://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html">https://www.easysoft.com/products/data_access/odbc-sql-server-driver/kerberos.html</a></p>
GSSLib = <i>value</i>	<p>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:</p> <pre>Krb5: failed to open gss lib (libgssapi_krb5.so)</pre> <p>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:</p> <pre>GSSLIB = /opt/extension/lib/libgssapi.so</pre>
GSSHost = Yes   No	<p>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.</p> <p>When turned on (set to Yes), the Easysoft ODBC-SQL Server Driver allows the use of GSS_C_NT_HOSTBASED_SERVICE.</p> <p>By default, GSSHOST is turned off (set to No).</p>
GSSFlag = <i>req_flags</i>	<p>The Easysoft ODBC-SQL Server Driver allows you to pass <i>req_flags</i> to the <code>gss_init_sec_context()</code> 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 <i>req_flags</i> to <code>gss_init_sec_context()</code>, use the GSSFLAG attribute:</p> <pre>GSSFLAG = <i>req_flags</i></pre> <p>where <i>req_flags</i> is a bitmask specifying the requested GSS services. To look up the available bitmask values, refer to the <code>gssapi.h</code> header file for the GSS-API distribution on the Easysoft ODBC-SQL Server Driver computer. The driver default GSSFLAG value is 4, which sets the GSS_C_REPLAY_FLAG flag.</p> <p>As an example, to request credential delegation, set the GSS_C_DELEG_FLAG flag by including this line in your data source:</p> <pre>GSSFLAG = 1</pre>

## 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	<p>When turned 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 <a href="#">SQLConnectW</a>, <a href="#">SQLDriverConnectW</a>, and <a href="#">SQLBrowseConnect</a>.</p> <p>The SQLServerUTF attribute must be specified in a section named ODBC in <code>odbc.ini</code>. For example:</p> <pre>[ ODBC ] SQLServerUTF = Yes</pre> <p>By default, SQLServerUTF is turned off (set to No).</p>

## Setting on Windows

The Easysoft ODBC-SQL Server Driver data source configuration dialog box, accessible when you create or edit an Easysoft ODBC-SQL Server Driver data source in **ODBC Data Source Administrator** lets you configure your data source.

For information about the data source attribute fields the dialog box contains, refer to the table [this topic](#).

## 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 computer name or IP address. Check that the computer 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 computer:

```
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 computer, "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 computer 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 get output similar to:

```
Connected to myserver  
Escape character is '^['
```

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

If you do not get 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're using the correct port but are unable to connect with telnet, the SQL Server instance may not allow remote TCP/IP connections. Refer to [Client unable to establish connection: Server not configured for TCP connection](#).

## 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 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 to find out what TCP port SQL Server is listening on. If the SQL Server Browser 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 computer, "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' error displays.

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 `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

Some applications allow you to make an ODBC connection without configuring a data source. To do this, you supply a connection string that contains the ODBC driver name and other driver-specific attribute-value pairs.

Here's an example Easysoft ODBC-SQL Server Driver connection string for Linux and UNIX:

```
DRIVER=Easysoft ODBC-SQL  
Server;Server=myhost\\SQLEXPRESS;UID=mydomain\\myuser;PWD=mypassword;Port=1500;Data  
base=Sales;
```

Here's an example Easysoft ODBC-SQL Server Driver connection string for Windows:

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

## Logging

If you report an issue to us, we may ask you to turn on ODBC Driver Manager or Easysoft ODBC-SQL Server Driver logging, to help us diagnose the cause of the issue.

To turn on logging, refer to the following sections.

**Note** If your application is a service (for example, Oracle or SQL Server), you may need to restart the service before enabling logging takes effect. To do this on Linux or UNIX, use `service`, `systemctl`, or a vendor-supplied script. To do this on Windows, use the Windows **Services** app.

## ODBC Driver Manager logging on Linux or UNIX

For the unixODBC Driver Manager, add the following attributes to the [ODBC] section (create one if none exists) in `odbcinst.ini`.

```
Trace = Yes
TraceFile = /path/filename
```

For example:

```
[ODBC]
Trace = Yes
TraceFile = /tmp/sql.log
```

Ensure that the user who's running the application to log has write permission to `TraceFile` (and to the directory containing it), otherwise no logging information will be produced.

## Easysoft ODBC-SQL Server Driver logging on Linux and UNIX

Driver manager trace files show all the ODBC calls an application makes, including their arguments and return values. Easysoft ODBC-SQL Server Driver logging is specific to the Easysoft driver and is of most use when making a support call.

To turn on Easysoft ODBC-SQL Server Driver logging, edit your ODBC data source in `odbc.ini`. For example:

```
[SQLSERVER_SAMPLE]
.
.
Logging = Yes
LogFile = /tmp/easysoft-odbc-driver.log
```

The value shown in the example specifies a log file named `/tmp/easysoft-odbc-driver.log`. Ensure that the user who's running the application to log has write permission to the log file (and to the directory containing it), otherwise no logging information will be produced.

## ODBC Driver Manager logging on Windows

1. In the Windows **Taskbar Search** box, enter "Run".
2. Do one of the following:
  - If your application is 64-bit, in the **Run** dialog box, enter:

```
odbcad32.exe
```

-Or-

- If your application is 32-bit, in the **Run** dialog box, enter:

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

**Note**

If you're not sure whether your application is 32-bit or 64-bit, start your application, then in Windows **Task Manager** check whether your application's process name contains (32-bit). For example, the process name for the 32-bit version of Excel is Microsoft Excel (32-bit); the process name for the 64-bit version of Excel is Microsoft Excel. On older versions of Windows, 32-bit applications contain \*32 in the process name rather than (32-bit). For applications such as Oracle or SQL Server that run as a service, check the \*Background processes\* list rather than the **Apps** list in **Task Manager**. If you're running a programming language from within a Windows command-line shell (for example, Command or PowerShell), in your shell, run the .exe file for the programming language. For example, run perl, php, python, or node. In **Task Manager**, expand the process list for **Windows Command Processor** or **Windows PowerShell**, as appropriate, and check whether the process for your programming language contains (32-bit).

3. Choose the **Tracing** tab.
4. Select **Machine-Wide tracing for all identities**.
5. Enter a log file name and path in the space provided. For example:

```
C:\Windows\Temp\SQL.log
```

6. Choose **Start Tracing Now**.

**Note** With SQL Server, you may get two Driver Manager log files, we need both. The first log file is in the folder that you specify in **ODBC Data Source Administrator**. The second file's location is defined by SQL Server. Two possible locations are the top-level folder (for example, C:\SQL.log) or the SQL Server temporary folder (for example, C:\Users\MSSQL\$SQLEXPRESS\AppData\Local\Temp\SQL.log). If the Driver Manager log file isn't in these folders, search for it on the drive where SQL Server is installed.

## Easysoft ODBC-SQL Server Driver logging on Windows

1. In the Windows **Taskbar Search** box, enter "Run".
2. Do one of the following:
  - If your application is 64-bit, in the **Run** dialog box, enter:

```
odbcad32.exe
```

-Or-

- If your application is 32-bit, in the **Run** dialog box, enter:

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

**Note** If you're not sure whether your application is 32-bit or 64-bit, start your application, then in Windows **Task Manager** check whether your application's process name contains (32-bit). For example, the process name for the 32-bit version of Excel is Microsoft Excel (32-bit); the process name for the 64-bit version of Excel is Microsoft Excel. On older versions of Windows, 32-bit applications contain \*32 in the process name rather than (32-bit). For applications such as Oracle or SQL Server that run as a service, check the \*Background processes\* list rather than the **Apps** list in **Task Manager**. If you're running a programming language from within a Windows command-line shell (for example, Command or PowerShell), in your shell, run the .exe file for the programming language. For example, run perl, php, python, or node. In **Task Manager**, expand the process list for **Windows Command Processor** or **Windows PowerShell**, as appropriate, and check whether the process for your programming language contains (32-bit).

3. Do one of the following:
  - If you configured a system data source, choose the **System DSN** tab.
  - Or-
  - If you configured a system data source, choose the **System DSN** tab.
4. Choose your Easysoft ODBC-SQL Server Driver data source from the list, and then choose **Configure**.
5. In the Easysoft ODBC-SQL Server Driver data source configuration dialog box, turn on **Driver Logging**.
6. Enter a log file name and path in the space provided. For example:

```
C:\Windows\Temp\Easysoft.log
```

## Finding out what product version you have on Windows

If you have an issue with the Easysoft ODBC-SQL Server Driver, we may ask you to tell us what your product version is. To find this out:

1. In the Windows **Taskbar**, enter “Add or remove programs” in the Windows **Search** box.
2. Select Easysoft ODBC-SQL Server Driver in the list.

The product version displays below.

## Client applications

How to work with SQL Server data in some example applications and programming languages:

- [Microsoft Access](#)
- [Microsoft Excel](#)
- [Microsoft Power BI](#)
- [Oracle](#)
- [LibreOffice](#)
- [Go](#)
- [Node.js](#)
- [Perl](#)
- [PHP](#)
- [Python](#)
- [R](#)

## Microsoft Access

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Access.
2. [Configure an ODBC data source](#).
3. Choose one of the following ways to work with your SQL Server data in Access.

### Linking a table

1. Open your Microsoft Access database.
2. Choose **External Data**.
3. In the **New Data Source** list, choose **From Other Sources > ODBC Database**.
4. In the **Get External Data** screen, choose **Link to the data source by creating a linked table**, and choose **OK**.
5. In the **Select Data Source** dialog box, choose the **Machine Data Source** tab.
6. Choose your Easysoft ODBC-SQL Server Driver ODBC data source from the **Machine Data Source** list, and then choose **OK**.
7. In the **Link Tables** dialog box, choose the tables that you want to link to, and then choose **OK**.

### Importing a table

1. Open your Microsoft Access database.
2. Choose **External Data**.
3. In the **New Data Source** list, choose **From Other Sources > ODBC Database**.
4. In the **Get External Data** screen, choose **Import the source data into a new table in the current database**, and choose **OK**.
5. In the **Select Data Source** dialog box, choose the **Machine Data Source** tab.
6. Choose your Easysoft ODBC-SQL Server Driver ODBC data source from the **Machine Data Source** list, and then choose **OK**.
7. In the **Import Objects** dialog box, choose the tables you want to import, and then choose **OK**.

## Microsoft Excel

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Excel.
2. [Configure an ODBC data source](#).
3. Choose one of the following ways to work with your SQL Server data in Excel.

### Data Connection Wizard

1. Choose **Data > Get Data > From Other Sources > From ODBC**.
2. Choose your Easysoft ODBC-SQL Server Driver data source from the list, and then choose **OK**.
3. Enter the user name and password for your data store if applicable, otherwise, enter any text string to get past this stage. Choose **Next**.
4. Choose the table that contains the data you want to retrieve, and then choose **Load**.

### Microsoft Query

1. Choose **Data > Get Data > From Other Sources > From Microsoft Query**.
2. In the **Choose Data Source** dialog box, choose your SQL Server data source from the list, and then choose **OK**.
3. In the **Query Wizard**, choose the columns that contain the data you want to retrieve, and then click **Next**.
4. If you want to return a subset of the data, use the **Filter Data** screen to filter the results of your query (this is the equivalent of a SQL WHERE clause), and then choose **Next**.
5. If you want to change the sort order of your data, use the **Sort Order** screen to sort the results of your query (this is the equivalent of a SQL ORDER BY clause), and then choose **Next**. Choose **Finish** to return your SQL Server data to Excel.

### PowerPivot

1. On the **PowerPivot** tab, choose **Manage**.
2. In the **PowerPivot** window, choose **Get External Data > From Other Sources**.
3. In the **Connect to a Data Source** list, choose **Others (OLEDB/ODBC)**.
4. In the **Specify a Connection** screen, enter a name for your connection in the space provided. Then choose **Build**.
5. In the **Data Link Properties** box, choose your Easysoft ODBC-SQL Server Driver data source from the list, and then choose **OK**.
6. Choose **Next**.
7. Choose how to import your SQL Server data and then choose **Finish**.
8. Choose **Close** to return the data to Excel.



## Microsoft Power BI

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Power BI Desktop.
2. [Configure an ODBC data source](#).
3. In Power BI Desktop, choose **Get data from another source**.
4. In the **Get Data** dialog box, choose **ODBC**, and then choose **Connect**.
5. In the **From ODBC** dialog box, choose your SQL Server data source, and then choose **OK**.
6. Enter your database user name and password when prompted.

If you make a mistake when entering the user name and password, cancel the connection process. Then in Power BI Desktop **Options and Settings**, edit the data source. Specify the correct user name or password in the data source credentials dialog box. Otherwise, Power BI Desktop will continue to use the cached incorrect credentials.

<b>Note</b>	If you do not normally need to enter a user name and password, enter some dummy strings in the spaces provided.
-------------	---

7. In the **Navigator** dialog box, choose the tables you want to analyse in Power BI Desktop, and then choose **Load**.

Your SQL Server data is now available to use in Power BI visualisations.

## Oracle

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Oracle.
2. [Configure an ODBC data source](#).
3. Follow the instructions for your Oracle platform.

## Connecting SQL Server to Oracle on Windows

1. Create a DG4ODBC init file on your Oracle machine. To do this, change to the %ORACLE\_HOME%\hs\admin directory. Create a copy of the file initdg4odbc.ora. Name the new file initMSSQL.ora.

**Note** In these instructions, replace %ORACLE\_HOME% with the location of your Oracle HOME directory. For example, C:\app\product\21c\homes\OraDB21Home1.

2. Ensure these parameters and values are present in your init file:

```
HS_FDS_CONNECT_INFO = "SQL Server"
```

Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data source.

3. Comment out the line that enables DG4ODBC tracing. For example:

```
#HS_FDS_TRACE_LEVEL = <trace_level>
```

4. Add an entry to %ORACLE\_HOME%\network\admin\listener.ora that creates a SID\_NAME for DG4ODBC. For example:

```
SID_LIST_LISTENER =
( SID_LIST =
( SID_DESC=
( SID_NAME=MSSQL )
( ORACLE_HOME=%ORACLE_HOME% )
( PROGRAM=dg4odbc )
)
)
```

5. Add a DG4ODBC entry to %ORACLE\_HOME%\network\admin\tnsnames.ora that specifies the SID\_NAME created in the previous step. For example:

```
MSSQL =
( DESCRIPTION =
( ADDRESS = ( PROTOCOL = TCP )( HOST = oracle_host )( PORT = 1521 ) )
( CONNECT_DATA =
( SID = MSSQL )
)
( HS = OK )
)
```

Replace oracle\_host with the host name of your Oracle machine.

6. Start (or restart) the Oracle Listener:

```
cd %ORACLE_HOME%\bin
lsnrctl stop
```

```
lsnrctl start
```

7. Connect to your Oracle database in SQL\*Plus.
8. In SQL\*Plus, create a database link for SQL Server. For example:

```
CREATE PUBLIC DATABASE LINK MSSQLLink
CONNECT TO "dbuser" IDENTIFIED BY "dbpassword"
USING 'MSSQL';
```

Replace dbuser and dbpassword with your backend user name and password, if applicable.

9. Try querying and updating your SQL Server data. For example:

```
SELECT "Surname" FROM "Customers"@MSSQLLink;

DECLARE
    num_rows integer;
BEGIN
    num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('INSERT INTO Customers (Surname, GivenName, City, Phone, CompanyName) VALUES
('Devlin'', 'Michaels'', 'Kingston'', '2015558966'', 'PowerGroup''));
END;
/

DECLARE
    num_rows integer;
BEGIN
    num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('UPDATE "Customers" SET "Surname" = ''Jones'' WHERE "CompanyName" =
''PowerGroup''');
END;
/

DECLARE
    num_rows integer;
BEGIN
    num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('DELETE from "Customers" WHERE CompanyName = ''PowerGroup'');
END;
/
```

## Notes

- If you have problems connecting to SQL Server from Oracle, enable DG4ODBC tracing and check the trace files written to the %ORACLE\_HOME%\hs\trace directory. To enable DG4ODBC tracing, add the line HS\_FDS\_TRACE\_LEVEL = DEBUG to initSQL Server.ora and then start or restart the Oracle listener. If the trace directory does not exist, create it.
- If you enable ODBC Driver Manager tracing, but do not get a log file in the location you specify, try looking in the top-level folder (for example, C:\SQL.log). Alternatively, in **ODBC Data Source Administrator**, change the trace file location to the Windows TEMP directory.

## Connecting SQL Server to Oracle on Linux and UNIX

1. Create a DG4ODBC init file on your Oracle machine. To do this, change to the \$ORACLE\_HOME\hs\admin directory. Create a copy of the file initdg4odbc.ora. Name the new file initMSSQL.ora.

**Note** In these instructions, replace \$ORACLE\_HOME with the location of your Oracle HOME directory. For example, /u01/app/oracle/product/21c/dbhome\_1.

2. Ensure these parameters and values are present in your init file:

```
HS_FDS_CONNECT_INFO = "SQL Server"
```

Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data source.

3. Comment out the line that enables DG4ODBC tracing. For example:

```
#HS_FDS_TRACE_LEVEL = <trace_level>
```

4. Add an entry to \$ORACLE\_HOME/network/admin/listener.ora that creates a SID\_NAME for DG4ODBC. For example:

```
SID_LIST_LISTENER =
(
  SID_LIST =
  (
    SID_DESC=
    (
      SID_NAME=MSSQL
      ORACLE_HOME=$ORACLE_HOME
      PROGRAM=dg4odbc
      ENV=LD_LIBRARY_PATH = /usr/local/easysoft/unixODBC/lib:
        /usr/local/easysoft/lib
    )
  )
)
```

Replace oracle\_home\_directory with the value of \$ORACLE\_HOME. For example, /u01/app/oracle/product/21c/dbhome\_1.

5. Add a DG4ODBC entry to \$ORACLE\_HOME/network/admin/tnsnames.ora that specifies the SID\_NAME created in the previous step. For example:

```
MSSQL =
(
  DESCRIPTION =
  (
    ADDRESS = (PROTOCOL = TCP)(HOST = oracle_host)(PORT = 1521)
    CONNECT_DATA =
    (
      SID = MSSQL
    )
  )
  (HS = OK)
)
```

Replace oracle\_host with the host name of your Oracle machine.

6. Start (or restart) the Oracle Listener:

```
cd $ORACLE_HOME/bin
./lsnrctl stop
./lsnrctl start
```

7. Connect to your Oracle database in SQL\*Plus.

8. In SQL\*Plus, create a database link for SQL Server. For example:

```
CREATE PUBLIC DATABASE LINK MSSQLLink
  CONNECT TO "dbuser" IDENTIFIED BY "dbpassword"
  USING 'MSSQL';
```

Replace dbuser and dbpassword with your backend user name and password, if applicable.

9. Try querying and updating your SQL Server data. For example:

```
SELECT "Surname" FROM "Customers"@MSSQLLink;

DECLARE
  num_rows integer;
BEGIN
  num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('INSERT INTO Customers (Surname, GivenName, City, Phone, CompanyName) VALUES
('Devlin', 'Michaels', 'Kingston', '2015558966', 'PowerGroup')));
END;
/

DECLARE
  num_rows integer;
BEGIN
  num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('UPDATE "Customers" SET "Surname" = ''Jones'' WHERE "CompanyName" =
''PowerGroup''');
END;
/

DECLARE
  num_rows integer;
BEGIN
  num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
('DELETE from "Customers" WHERE CompanyName = ''PowerGroup''');
END;
/
```

## Notes

- If you have problems connecting to SQL Server from Oracle, enable DG4ODBC tracing and check the trace files written to the \$ORACLE\_HOME/hs/trace directory. To enable DG4ODBC tracing, add the line HS\_FDS\_TRACE\_LEVEL = DEBUG to initSQL Server.ora and then start or restart the Oracle listener. If the trace directory does not exist, create it.

## LibreOffice

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as LibreOffice.
2. [Configure an ODBC data source](#).
3. Choose **File > New > Database**.
4. Choose **Connect to an existing database**.
5. Choose **ODBC** in the list, and then choose **Next**.
6. Choose **Browse**, double-click your data source, and then choose **Next**.
7. If your database requires a database user name, enter it in the **User name** box. If this user needs to supply a password choose the **Password required** check box.
8. Choose **Finish**.
9. Save the database when prompted.

The database opens in a new Base window. From here you can access your data.

10. In the left pane of the database window, choose the **Tables** icon to display a hierarchy of tables. Enter the database password if prompted, and then choose **OK**.
11. To retrieve the data in a table, in the **Tables** pane, double-click a table.
12. Choose the **Queries** icon to create a query.

Use any of the methods listed in the **Tasks** pane to create a query.

## Go

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Go.
2. [Configure an ODBC data source](#).
3. Install the `odbc` package for Go:

```
go mod init test
go get github.com/alexbrainman/odbc
```

4. Create and then use Go to run this script, which retrieves some SQL Server data:

```
package main

import (
    _ "github.com/alexbrainman/odbc"
    "database/sql"
    "log"
)

func main() {
    // Replace the DSN value with the name of your ODBC data source.
    db, err := sql.Open("odbc",
        "DSN=SQL Server")
    if err != nil {
        log.Fatal(err)
    }

    var (
        name string
    )

    rows, err := db.Query("SELECT Surname FROM Customers")
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()
    for rows.Next() {
        err := rows.Scan(&name)
        if err != nil {
            log.Fatal(err)
        }
        log.Println(name)
    }
    err = rows.Err()
    if err != nil {
        log.Fatal(err)
    }

    defer db.Close()
}
```

## Node.js

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Node.js.
2. [Configure an ODBC data source](#).
3. Install the `odbc` module for Node.js:

```
npm install odbc
```

4. Create and then use Node.js to run this script, which retrieves some SQL Server data:

```
const odbc = require('odbc');
// Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver
// data source.
const connection = odbc.connect('DSN=SQL Server', (error, connection) => {
  connection.query('SELECT Surname FROM Customers', (error, result) => {
    if (error) { console.error(error) }
    console.log(result);
  });
});
```

5. This script retrieves the tables and views in your Easysoft ODBC-SQL Server Driver data source:

```
const odbc = require('odbc');
const connection = odbc.connect('DSN=SQL Server', (error, connection) => {
  connection.tables(null, null, null, null, (error, result) => {
    if (error) { return; }
    const util = require('util');
    console.log(util.inspect(result, {maxLength: null, depth:null}))
  });
});
```

6. This script retrieves the names of the columns in these tables and views:

```
const odbc = require('odbc');
const connection = odbc.connect('DSN=SQL Server', (error, connection) => {
  connection.columns(null, null, null, null, (error, result) => {
    if (error) { return; }
    const util = require('util');
    console.log(util.inspect(result, {maxLength: null, depth:null}))
  });
});
```

7. These scripts insert, update, and then delete some SQL Server data:

```
const odbc = require("odbc");
const connection = odbc.connect("DSN=SQL Server", (error, connection) => {
  connection.query("INSERT INTO
Customers (
  Surname,
  GivenName,
  City,
  Phone,
  CompanyName
```



```

    )
VALUES
    (
        'Devlin',
        'Michaels',
        'Kingston',
        '2015558966',
        'PowerGroup'
    ), (error, result) => {
        if (error) { console.error(error) }
        console.log(result);
    });
});

const odbc = require("odbc");
const connection = odbc.connect("DSN=SQL Server", (error, connection) => {
    connection.query("UPDATE Customers SET Surname = 'Jones' WHERE CompanyName =
'PowerGroup'", (error, result) => {
        if (error) { console.error(error) }
        console.log(result);
    });
});

const odbc = require("odbc");
const connection = odbc.connect("DSN=SQL Server", (error, connection) => {
    connection.query("DELETE FROM Customers WHERE CompanyName = 'PowerGroup'",
(error, result) => {
        if (error) { console.error(error) }
        console.log(result);
    });
});
});

```

## Perl

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Perl.
2. [Configure an ODBC data source](#).
3. Check whether your Perl distribution supports ODBC:

```
perl -e 'use DBD::ODBC;'
```

4. Do one of the following:
  - If you get no output, your Perl distribution supports ODBC. Skip to the next step.
  - If you get:

```
Can't locate DBD/ODBC.pm
```

you need to [install DBD::ODBC](#) before you can use the Easysoft ODBC-SQL Server Driver to connect to SQL Server.

5. Create and then use Perl to run this script, which retrieves some SQL Server data:

```
use strict;
use DBI;
# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sql = "SELECT Surname FROM Customers";

my $sth = $dbh->prepare($sql)
    or die "Can't prepare statement: $DBI::errstr";

$sth->execute();

my($Col);

# Fetch and display the result set values.
while(($Col) = $sth->fetchrow()){
    print("$Col\n");
}

$dbh->disconnect if ($dbh);
```

6. This script retrieves the tables and views in your Easysoft ODBC-SQL Server Driver data source:

```
use strict;
use DBI;
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sth = $dbh->table_info()
    or die "Can't prepare statement: $DBI::errstr";

my @row;

while (@row = $sth->fetchrow_array) {
```

```

        print join(", ", @row), "\n";
    }

    $dbh->disconnect if ($dbh);

```

7. This script retrieves the names of the columns in these tables and views:

```

use strict;
use DBI;
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sth = $dbh->column_info('', '', '', '')
    or die "Can't prepare statement: $DBI::errstr";

my @row;
while (@row = $sth->fetchrow_array) {
    print join(", ", @row), "\n";
}

$dbh->disconnect if ($dbh);

```

8. These scripts insert, update, and then delete some SQL Server data:

```

use strict;
use DBI;
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sth = $dbh->prepare(q/INSERT INTO Customers (Surname, GivenName, City, Phone,
CompanyName) VALUES (?, ?, ?, ?, ?)/)
    or die "Can't prepare statement: $DBI::errstr";

$sth->execute('Devlin', 'Michaels', 'Kingston', '2015558966', 'PowerGroup');

$dbh->disconnect if ($dbh);

use strict;
use DBI;
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sth = $dbh->prepare('UPDATE Customers SET Surname = \'Jones\' WHERE
CompanyName = ?')
    or die "Can't prepare statement: $DBI::errstr";

$sth->execute('PowerGroup');

$dbh->disconnect if ($dbh);

use strict;
use DBI;
my $dbh = DBI-> connect('dbi:ODBC:SQL Server');

my $sth = $dbh->prepare('DELETE FROM Customers WHERE CompanyName = ?')

```

```
        or die "Can't prepare statement: $DBI::errstr";

$sth->execute('PowerGroup');

$dbh->disconnect if ($dbh);
```

### Further information

- [Perl DBI DBD::ODBC tutorial: Drivers, data sources, and connection](#)

## PHP

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as PHP.
2. [Configure an ODBC data source](#).
3. Check whether your PHP distribution supports ODBC. In php.ini, make sure there is no comment character (;) before the extension\_dir and extension=odbc settings (;extension\_dir=directory becomes extension\_dir=directory and ;extension=odbc becomes extension=odbc).
4. Create and then use PHP to run this script, which retrieves some SQL Server data:

```
<?php
// Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver
data source.
// If your database requires a user name and password, supply them in the
odbc_connect_call.
$con = odbc_connect("SQL Server", "", "");
$stmt = odbc_exec($con, "SELECT * FROM Customers");
// You may need to change the capitalisation of Surname to all upper case or
all lower case.
while ($row = odbc_fetch_array($stmt)) {
    echo "Surname = ", $row["Surname"], "\n";
}
odbc_close($con);
?>
```

5. This script retrieves the tables and views in your Easysoft ODBC-SQL Server Driver data source:

```
<?php
$con = odbc_connect("SQL Server", "", "");
$tables = odbc_tables($con);
while (($row = odbc_fetch_array($tables))) {
    print_r($row);
}
odbc_close($con);
?>
```

6. This script retrieves the names of the columns in these tables and views:

```
<?php
$con = odbc_connect("SQL Server", "", "");
$columns = odbc_columns($con);
while (($row = odbc_fetch_array($columns))) {
    print_r($row);
}
odbc_close($con);
?>
```

7. These scripts insert, update, and then delete some SQL Server data:

```
<?php
$cnx = odbc_connect("SQL Server", "", "");
$stmt = odbc_prepare($cnx, "INSERT INTO Customers (Surname, GivenName, City,
Phone, CompanyName) VALUES (?, ?, ?, ?, ?)");
```

```
$success = odbc_execute($stmt, array('Devlin', 'Michaels', 'Kingston',  
'2015558966', 'PowerGroup'));  
odbc_close($cnx);  
?>  
  
<?php  
$cnx = odbc_connect("SQL Server", "", "");  
$stmt = odbc_prepare($cnx, "UPDATE Customers SET Surname = 'Jones' WHERE  
CompanyName = ?");  
$success = odbc_execute($stmt, array('PowerGroup'));  
odbc_close($cnx);  
?>  
  
<?php  
$cnx = odbc_connect("SQL Server", "", "");  
$stmt = odbc_prepare($cnx, "DELETE FROM Customers WHERE CompanyName = ?");  
$success = odbc_execute($stmt, array('PowerGroup'));  
odbc_close($cnx);  
?>
```

### Further information

- [Easysoft PHP tutorials and code samples](#)

# Python

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as Python.
2. [Configure an ODBC data source](#).
3. Check whether your Python distribution supports ODBC.

```
pip list
```

If you don't have pip installed:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
```

4. Do one of the following:
  - If the output contains pyodbc, your Python distribution supports ODBC. Skip to the next step.
  - If the output does not contain pyodbc, use pip to install this module:

```
pip install pyodbc
```

5. Create and then use Python to run this script, which retrieves some SQL Server data:

```
import pyodbc

# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
sql = "SELECT Surname FROM Customers"
cursor.execute(sql)
rows = cursor.fetchall()
# You may need to change the capitalisation of Surname to all upper case or all
lower case.
for row in rows:
    print(row.Surname)
exit()
```

6. This script retrieves the tables and views in your Easysoft ODBC-SQL Server Driver data source:

```
import pyodbc

# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
cursor.tables()
rows = cursor.fetchall()
for row in rows:
    print(row.table_name)
exit()
```

7. This script retrieves the names of the columns in these tables and views:

```
import pyodbc
```

```
# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
cursor.columns()
rows = cursor.fetchall()
for row in rows:
    print(row.table_name, row.column_name)
exit()
```

8. These scripts insert, update, and then delete some SQL Server data:

```
import pyodbc

cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
sql = "INSERT INTO Customers (Surname, GivenName, City, Phone, CompanyName)
VALUES (?, ?, ?, ?, ?)"
cursor.execute(sql, 'Devlin', 'Michaels', 'Kingston', '2015558966', 'PowerGroup')
cursor.commit()
exit()
```

```
import pyodbc

cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
sql = "UPDATE Customers SET Surname = 'Jones' WHERE CompanyName = ?"
cursor.execute(sql, 'PowerGroup')
cursor.commit()
exit()
```

```
import pyodbc

cnxn = pyodbc.connect("DSN=SQL Server")
cursor = cnxn.cursor()
sql = "DELETE FROM Customers WHERE CompanyName = ?"
cursor.execute(sql, 'PowerGroup')
cursor.commit()
exit()
```

### Further information

- [Easysoft Python tutorials and code samples](#)



## R

1. [Install the Easysoft ODBC-SQL Server Driver](#) on same computer as R.
2. [Configure an ODBC data source](#).
3. In R Console, check whether your R distribution supports ODBC.

```
library("RODBC")
```

4. Do one of the following:
  - If you get no output, you have the ODBC library for R. Skip to the next step.
  - If you get an "there is no package" error, install the ODBC library for R:

```
install.packages("RODBC")
```

5. Create and then use R to run this script, which retrieves some SQL Server data:

```
library("RODBC")
# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
ch <- odbcConnect("SQL Server")
sqlQuery(ch, paste("SELECT Surname FROM Customers"))
odbcClose(ch)
quit()
```

6. This script retrieves the tables and views in your Easysoft ODBC-SQL Server Driver data source:

```
library("RODBC")
# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
ch <- odbcConnect("SQL Server")
sqlTables(ch)
odbcClose(ch)
quit()
```

7. This script retrieves the names of the columns in the specified table or view:

```
library("RODBC")
# Replace SQL Server with the name of your Easysoft ODBC-SQL Server Driver data
source.
ch <- odbcConnect("SQL Server")
# You may need to change the capitalisation of Customers to all upper case or all
lower case.
sqlColumns(ch, sqtable="Customers")
odbcClose(ch)
quit()
```

8. These scripts insert, update, and then delete some SQL Server data:

```
library("RODBC")
ch <- odbcConnect("SQL Server")
sqlQuery(ch, paste("INSERT INTO Customers (Surname, GivenName, City, Phone,
CompanyName) VALUES ('Devlin', 'Michaels', 'Kingston', '2015558966',
'PowerGroup')"))
```

```
odbcClose(ch)
quit()

library("RODBC")
ch <- odbcConnect("SQL Server")
sqlQuery(ch, paste("UPDATE Customers SET Surname = 'Jones' WHERE CompanyName =
'PowerGroup'"))
odbcClose(ch)
quit()

library("RODBC")
ch <- odbcConnect("SQL Server")
sqlQuery(ch, paste("DELETE FROM Customers WHERE CompanyName = 'PowerGroup'"))
odbcClose(ch)
quit()
```

## About the Easysoft ODBC-SQL Server Driver

The Easysoft ODBC-SQL Server Driver provides real-time access to SQL Server data from any application that supports ODBC.

In this section:

- [ODBC API functions](#)
- [Cursor support](#)
- [Supported data types](#)
- [Unicode support](#)
- [ANSI-Only version of the Easysoft ODBC-SQL Server Driver](#)
- [The XML data type](#)
- [Using large-value data types](#)
- [Snapshot isolation](#)
- [Performing bulk copy operations](#)
- [Table-valued parameters](#)
- [Binding procedure parameters by name](#)
- [SQL Server authentication modes](#)
- [Encrypting connections to SQL Server](#)
- [Database mirroring](#)
- [Connection failover](#)
- [Connecting to SQL Server by using IPv6](#)
- [Example SQL statements](#)

## ODBC API functions

Use this table to find out what ODBC API functions the Easysoft ODBC-SQL Server Driver supports:

Function	Status
SQLAllocConnect	Supported
SQLAllocEnv	Supported
SQLAllocHandle	Supported
SQLAllocStmt	Supported
SQLBindCol	Supported
SQLBindParameter	Supported
SQLBrowseConnect	Not supported
SQLBulkOperations	Supported
SQLCancel	Supported
SQLCloseCursor	Supported
SQLColAttribute	Supported
SQLColAttributes	Supported
SQLColumnPrivileges	Supported
SQLColumns	Supported
SQLConnect	Supported
SQLCopyDesc	Supported
SQLDisconnect	Supported
SQLDriverConnect	Supported
SQLDrivers	Supported
SQLEndTran	Supported
SQLError	Supported
SQLExecDirect	Supported
SQLExecute	Supported
SQLExtendedFetch	Supported
SQLFetch	Supported
SQLFetchScroll	Supported
SQLForeignKeys	Supported
SQLFreeConnect	Supported
SQLFreeEnv	Supported
SQLFreeHandle	Supported
SQLFreeStmt	Supported
SQLGetConnectAtt	Supported
SQLGetConnectOption	Supported

Function	Status
SQLGetCursorName	Supported
SQLGetData	Supported
SQLGetDescField	Supported
SQLGetDescRec	Supported
SQLGetDiagField	Supported
SQLGetDiagRec	Supported
SQLGetEnvAttr	Supported
SQLGetFunctions	Supported
SQLGetInfo	Supported
SQLGetStmtAttr	Supported
SQLGetStmtOption	Supported
SQLGetTypeInfo	Supported
SQLMoreResults	Supported
SQLNativeSql	Supported
SQLNumParams	Supported
SQLNumResultCols	Supported
SQLParamData	Supported
SQLParamOptions	Supported
SQLPrepare	Supported
SQLPrimaryKeys	Supported
SQLProcedureColumns	Supported
SQLProcedures	Supported
SQLPutData	Supported
SQLRowCount	Supported
SQLSetConnectAttr	Supported
SQLSetConnectOption	Supported
SQLSetCursorName	Supported
SQLSetDescField	Supported
SQLSetDescRec	Supported
SQLSetEnvAttr	Supported
SQLSetParam	Supported
SQLSetPos	Supported
SQLSetScrollOptions	Supported
SQLSetStmtOption	Supported
SQLSetStmtAttr	Supported

Function	Status
SQLStatistics	Supported
SQLTablePrivileges	Supported
SQLTables	Supported
SQLTransact	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

<b>Note</b>	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 an 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 <sqlncli.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 computer, 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=mywindowpassword",
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.)

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 [snapshot isolation](#).

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 querying.

## 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.

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

In this C code sample, deferred statement preparation is turned off. 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)

## 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 [Easysoft ODBC-SQL Server] 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

The XML data type lets you store 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/adventure-
works/ProductModelManuInstructions" LaborHours="2.5"...LocationID="10">
```

The next 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

The MAX specifier 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

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 turned 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 SQL Server-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] [-O] -useNOSSL
```

The arguments are

**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 mycomputer\\sqlexpress.

**-D logfile**

Turns on 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's 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\_IDENTIFIER 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 computer to be established before returning to the bcp.

-G [req\_flags]

Set the GSSFlag attribute via bcp.

-O

Connect by using an ODBC connection string. For example:

```
bcp myTable out my.dat -O "DSN=SQLSERVER_SAMPLE".
```

-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 computer 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 computer. For more information about kinit and accessing SQL Server as a Kerberos service, refer to the following Easysoft tutorial:

[https://www.easysoft.com/products/data\\_access/odbc-sql-server-driver/kerberos.html](https://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 "mysqlservercomputer\myinstance" -K -M
MSSQLSvc/mysqlservercomputer: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

## -useNOSSL

You should not normally need to include this flag.

# Easysoft ODBC-SQL Server Driver extensions - bulk copy functions

In this section:

- [bcp\\_batch](#)
- [bcp\\_bind](#)
- [bcp\\_colfmt](#)
- [bcp\\_collen](#)
- [bcp\\_colptr](#)
- [bcp\\_columns](#)
- [bcp\\_control](#)
- [bcp\\_done](#)
- [bcp\\_exec](#)
- [bcp\\_getcolfmt](#)
- [bcp\\_gettypename](#)
- [bcp\\_init](#)
- [bcp\\_moretext](#)
- [bcp\\_readfmt](#)
- [bcp\\_sendrow](#)
- [bcp\\_setbulkmode](#)

- [bcp\\_setcolfmt](#)
- [bcp\\_writfmt](#)

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, 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 eUserDataTypes,  
INT cbIndicator,  
DBINT cbUserData,  
LPCBYTE pUserDataTerm,  
INT cbUserDataTerm,  
INT idxServerCol);
```

## Arguments

`hdbc`

The bulk copy-enabled ODBC connection handle.

`idxUserDataCol`

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

`eUserDataTypes`

The data type of this column in the user file.

The `eUserDataTypes` 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`

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`

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

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

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\_colln

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

### Syntax

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

### Arguments

hdbc

The bulk copy-enabled ODBC connection handle.

cbData

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

The bulk copy-enabled ODBC connection handle.

pData

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

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

The bulk copy-enabled ODBC connection handle.

nColumns

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.

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`

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, causes `bcp_readfmt` to read at execution.

**BCPFILEFMT**

The version number of the data file format.

**BCPFIRST**

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, 2025 is converted to the character string: `2025-01-01 00:00:00.000`. When `TRUE`, the same datetime value is represented as: `{ts '2025-01-01 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_writelfmt(hdbc1, "myFmtFile.fmt"));
    if ((retcode != SUCCEED)) {
        printf("bcp_writelfmt(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`

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`

The bulk copy-enabled ODBC connection handle.

`pnRowsProcessed`

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 <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_writelfmt(hdbc1, "myFmtFile.fmt"));
if ((retcode != SUCCEED)) {
    printf("bcp_writelfmt(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\_getcolfmt

Used to find the column format property value.

### Syntax

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

### Arguments

hdbc

The bulk copy-enabled ODBC connection handle.

field

The column number for which the property is retrieved.

property

One of the property constants.

pValue

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_writelfmt(hdbc1, "myFmtFile.fmt"));
if ((retcode != SUCCEED)) {
    printf("bcp_writelfmt(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,
    DBB00L 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

The bulk copy-enabled ODBC connection handle.

szTable

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

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

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`

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`

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

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.

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

The bulk copy-enabled ODBC connection handle.

property

A constant of type BYTE. Refer to 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

The bulk copy-enabled ODBC connection handle.

field

The ordinal column number for which the property is being set.

property

One of the property constants. Property constants are defined in this table

Value	Description
BCP_FMT_TYPE	<p>BYTE</p> <p>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.</p> <p>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.</p> <p>To specify the default data representation for the SQL Server data type, set this parameter to 0.</p>
BCP_FMT_DATA_LEN	<p>DBINT</p> <p>The length in bytes of the data (column length).</p>
BCP_FMT_TERMINATOR	<p>LPCBYTE</p> <p>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.</p>
BCP_FMT_SERVER_COLUMN	<p>INT</p> <p>Ordinal position of the column in the database</p>
BCP_FMT_COLLATION	<p>LPCSTR</p> <p>Collation name.</p>

pValue

The pointer to the value to associate to the property. It allows each column format property to be set individually.

cbValue

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 don't 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\_COLUMN 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

The bulk copy-enabled ODBC connection handle.

szFormatFile

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, 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.TVPIItem (
* 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 TVPIItem (OrdNo, ProdCode, Qty)
* SELECT @OrdNo, ProdCode, Qty FROM @Items
*/

SQLDriverConnect(dbc, NULL, "DSN=SQLSERVER_SAMPLE;", 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%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: IS09000

@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. The ways that SQL Server provides to authenticate SQL Server logins include: 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 turned 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, refer to `/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/libstdscrypt.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 than that used for trusted connections. The SQL Server user name is sent in plain text.

## Encrypting connections to SQL Server

SQL Server can 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 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](#)
- [Configuring and testing SSL encryption](#)
- [Encrypting the login packet](#)
- [Easysoft ODBC-SQL Server Driver with SSL Support attribute fields](#)

## 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](#) for information about setting up SSL encryption on the client and SQL Server computer.

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?
- If neither of the preceding points are true, where on the Easysoft ODBC-SQL Server Driver with SSL Support computer 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 = MYSQLSERVERCOMPUTER\MYINSTANCE
Port =
Database =
User = MYDOMAIN\myuser
Password = mypassword
```

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 computer. This information is intended to supplement rather than replace the Microsoft SQL Server documentation.

### Installing an SSL certificate

Before you can access SQL Server over an encrypted connection, an SSL certificate needs to be installed on the SQL Server computer. SQL Server can use an SSL certificate from a trusted CA if available or generate a self-signed certificate.

**Note** Even though SQL Server 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.

Refer to the following Microsoft documentation for installation information:

<https://msdn.microsoft.com/en-us/library/ms189067.aspx>

### Testing that SSL is available on the SQL Server computer

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. 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.

**Note** 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.

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 computer, the client computer 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 computer rather than the SQL Server computer, 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 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 computer 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 computer 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 computer:

1. On the SQL Server computer, in the Windows **Run** dialog box, enter:

```
mmc
```

2. In Microsoft **Management Console**, on the **File** menu, choose **Add/Remove Snap-in**, and then choose **Add**.
3. In the **Add Standalone Snap-In** dialog box, double-click **Certificates**. Choose **Computer account** when prompted and then choose **Next**. Choose **Finish**.
4. Choose **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 choose **Certificates**. In the right pane of the console window, right-click the **CA** certificate, point to **All Tasks**, and then choose **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 computer 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 computer or the client computer. If you do not want to enable encryption globally on the SQL Server computer, you can enable encryption on a per-client basis.

To configure the client to request an encrypted connection:

1. 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 choose **Properties**. Make sure that **ForceEncryption** is set to **No**.
2. On the computer where the Easysoft ODBC-SQL Server Driver with SSL Support is installed, create an ODBC data source to connect to the SQL Server instance.
3. 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 = MYSQLSERVERCOMPUTER\MYINSTANCE
User = MYDOMAIN\myuser
Password = mypassword
```

```
Encrypt = No
TrustServerCertificate = No
```

4. Use an application such as Microsoft Network Monitor, Snort, or Ethereal to capture network traffic between this computer and the SQL Server computer.

Using Network Monitor or a network sniffer tool lets you verify that you have successfully made an encrypted connection to the SQL Server computer. When testing the Easysoft driver, we used Network Monitor on the SQL Server computer and Snort on the client computer to capture network traffic.

5. 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
```

6. 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 computer 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.....
```

7. Press RETURN to exit isql.
8. In your data source, set the Encrypt attribute to Yes.
9. Do one of the following:
  - To connect to a SQL Server instance on a computer 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 computer, 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 computer and copied it to this computer, specify the path to the certificate file. For example, CertificateFile = /usr/share/ssl/CA/MyCA.cer.
  - To connect to a SQL Server instance that is using a self-signed certificate, set the TrustServerCertificate attributes to Yes.
10. Use isql to connect to the data source and retrieve the same data as you did in step 4.
11. 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..0.
```

```
&...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.V0r.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 uses 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 computer.

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   Strict	<p>Whether the client requests an encrypted connection to SQL Server.</p> <p>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.</p> <p>Do not enable SSL encryption on both the server and client, use one or the other.</p> <p>If you need to connect to SQL Server with strict encryption, set to Strict. Strict encryption was introduced in SQL Server 2022. In Strict encryption mode, TrustServerCertificate is ignored, the server certificate is checked and data sent between client and server is encrypted. (Refer also to the HostNameInCertificate attribute.)</p>

Attribute	Description
EncryptDeny = Yes   No	Whether the client denies it supports encryption. You should not normally need to use this attribute. The attribute was added as a workaround for a customer who was getting the error "Client unable to establish connection: required SSL (failed to receive packet)".
HostNameInCertificate = <i>value</i>	Specifies the hostname to be expected in the server certificate when encryption is negotiated. Use HostNameInCertificate if this hostname is different from the one specified with the Server attribute.
TrustServerCertificate = Yes   No	<p>Enables the client to request encryption even when an SSL certificate has not been installed on the SQL Server computer.</p> <p>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 computer. 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.</p> <p>If the SQL Server <b>ForceEncryption</b> option is enabled, the TrustServerCertificate value is ignored. When encryption is enabled on the SQL Server computer, the Easysoft ODBC-SQL Server Driver with SSL Support bypasses the validation of the server certificate.</p> <p>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.</p> <p>By default, TrustServerCertificate is turned on (set to Yes).</p>
CertificateFile = <i>filename</i>	<p>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.</p> <p>If the CA certificate is not installed on your client computer, you need to export the certificate on the SQL Server computer and install it on the client.</p> <p><b>Examples</b></p> <p>To load a CA certificate from the root CA certificate store supplied with the OpenSSL distribution, use:</p> <pre>CertificateFile = /usr/share/ssl/certs/ca-bundle.crt</pre> <p>To load a private CA certificate named MyCA.cer that you copied to /usr/share/ssl/CA, use:</p> <pre>CertificateFile = /usr/share/ssl/CA/MyCA.cer</pre>



Attribute	Description
Cypher = <i>value</i>	<p>The cypher suite that the Easysoft ODBC-SQL Server Driver with SSL Support will request during the SSL handshake with the SQL Server computer.</p> <p>A cypher suite is a set of authentication, encryption, and data integrity algorithms used to protect data exchanged between computers. During the SSL handshake part of the connection process, the SSL layer in the ODBC driver and the Schannel layer on the SQL Server computer negotiate to decide which cypher suite they will use.</p> <p>To find out 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:</p> <pre>LOGFILE = /tmp/sql-server-driver.log LOGGING = Yes</pre> <p>Connect and then examine the driver log file. Look for a log file entry similar to:</p> <pre>SSL using cypher 'RC4-MD5 SSLv3 Kx=RSA Au=RSA Enc=RC4(128) Mac=MD5'</pre> <p>This entry shows that the ODBC driver and the SQL Server computer negotiated the following cryptographic protection for the connection:</p> <pre>Encryption: RC4 Encryption strength: 128-bit Cryptographic checksum: MD5 Authentication: RSA</pre> <p>(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, refer to <a href="https://support.microsoft.com/kb/260729">https://support.microsoft.com/kb/260729</a>.)</p> <p>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:</p> <pre># Request Triple DES (3DES) for data encryption. # Request Secure Hash Algorithm (SHA) for data # integrity protection. Cypher = 3DES+SHA</pre> <p>-Or-</p> <pre># Request Advanced Encryption Standard (AES) for data # encryption. If AES is not available on the server, # request 3DES. Cypher = AES:3DES</pre>



Attribute	Description
Cypher = <i>value</i>	<p data-bbox="456 161 507 192">-Or-</p> <div data-bbox="475 228 1477 412"> <pre># Use Secure Hash Algorithm (SHA) to protect data # integrity. Let the SSL layers negotiate which # encryption algorithm to use. Cypher = SHA</pre> </div> <p data-bbox="456 425 1490 542">If you specify a cypher suite that is not available on the server computer, the Easysoft ODBC-SQL Server Driver with SSL Support returns the error "Required SSL (failed to receive packet)".</p> <p data-bbox="456 555 1452 672">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".</p> <p data-bbox="456 685 1187 766">(For a complete list of valid Cypher values, refer to <a href="https://www.openssl.org/docs/apps/ciphers.html">https://www.openssl.org/docs/apps/ciphers.html</a>.)</p> <p data-bbox="456 779 1477 936">Note that if you're 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.</p> <p data-bbox="456 949 1471 1106">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, refer to <a href="https://support.microsoft.com/kb/920995">https://support.microsoft.com/kb/920995</a>.</p>
Entropy = <i>filename</i>	<p data-bbox="456 1128 1490 1357">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.</p> <p data-bbox="456 1370 1468 1487">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)".</p> <p data-bbox="456 1500 1477 1738">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:</p> <div data-bbox="475 1774 1477 1832"> <pre>Entropy = /etc/entropy</pre> </div>

## Database mirroring

Database mirroring 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](#)
- [Data source attributes for a mirrored database](#)
- [Connection retry algorithm](#)
- [The impact of a stale failover partner name](#)

## 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 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, refer to [Data source attributes for a mirrored database](#).

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 computers, such as when a client computer 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, refer to [Connection attributes](#).

Attribute	Notes
Server	<p>Use the Server attribute to specify the current principal database server. You can use the following format for the attribute value: <code>computername[instancename]</code>. For example:</p> <pre>Server=partner_A_host</pre> <p>-Or-</p> <pre>Server=partner_A_host\instance_2</pre> <p>Note that when you specify a computer 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: <code>Server=ip_address:port</code> (IPv4 format) or <code>Server=ip_address*port</code> (IPv6 format). If you specify an IPv6 address, you also need to set the IPv6 attribute to 1. For example:</p> <pre># IPv4 address Server = 123.34.45.56:4724</pre> <p>-Or-</p> <pre># IPv6 address Server = 2001:4898:23:1002:20f:1fff:feff:b3a3*7022 IPv6    = 1</pre>

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	<p>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.</p> <p>The Failover_Partner attribute value takes the form <code>computername[instancename]</code>. For example:</p> <pre>Server=partner_B_host\instance_2</pre> <p>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.</p> <p>To find out the current failover partner for a mirrored database, use <code>tdshelper</code>. For information about how to do this, refer to the <a href="#">The impact of a stale failover partner name</a>.</p>
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.

## 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 computer 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_mirroredb -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_mirroredb'.
Connection: connected to my_initial_partner as myuser with mypassword,
database='my_mirroredb', partner='my_failover_partner'
```

## Connection failover

Connection failover maintains data availability by allowing an application to connect to a backup SQL Server computer 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 turned on, 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 computer 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, refer to [Connection attributes](#).

## Connecting to SQL Server 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.

The Easysoft ODBC-SQL Server Driver supports both IPv4 and IPv6.

## Configuring your client computer for IPv6

IPv6 needs to be enabled on the client computer. The procedure for this depends on the client platform. For more information, consult the IPv6 documentation for your system. For Linux systems, consult <https://www.tldp.org/HOWTO/Linux+IPv6-HOWTO/>.

## Configuring your ODBC data source for IPv6

To connect to a SQL Server instance that's 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 computer's host name. Otherwise, specify its IPv6 address. For example:

```
Server=myipv6computer\myinstance
IPv6 = 1
```

-Or-

```
Server=ABCD:EF12:0:0:0:0:3456\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.

## Finding out more about data types on Windows

If you need more information about a data types, for example, the precision and scale, use Microsoft's ODBC Test to do this.

1. Download the version of ODBC Test that matches your application's architecture from:

<https://www.easysoft.com/ftp/pub/utils/windows/odbc-test/>

2. Copy both files to a folder on the machine where Easysoft ODBC-SQL Server Driver is installed.
3. Double-click **odbcte32.exe**.

4. Select **Con > Full Connect**.
5. Choose your Easysoft ODBC-SQL Server Driver data source from the list.
6. Choose **Catalog > SQLGetTypeInfo**.
7. Either choose **SQL\_ALL\_TYPES=0 (1.0)** or a specific data type from the **DataType** list.
8. Choose **Results > Get Data All**.



## Example SQL statements

### Example queries

- To fetch all records from a table, use the asterisk symbol (\*) in your queries. For example:

```
SELECT * FROM Customers
```

- To only fetch records whose values are different, use DISTINCT. For example:

```
-- Which different sales regions are there?  
SELECT DISTINCT Region AS Different_Regions FROM SalesOrders  
-- How many different sales regions are there?  
SELECT COUNT(DISTINCT Region) AS Different_Regions FROM SalesOrders
```

- To filter records, use WHERE. For example:

```
SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region = 'Eastern'

SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region = 'Eastern'
    OR Region = 'Western'

SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region = 'Eastern'
    AND EXTRACT(YEAR FROM OrderDate) = 2025
```

You can also supply a WHERE clause value as a parameter. For example, to do this in [Python](#):

```
cursor.execute("SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region = ?", ['Eastern'])
```

- To fetch records that don't match the WHERE clause pattern use NOT. For example:

```
SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    NOT Region = 'Eastern'
```

- To sort the result set in either ascending or descending order, use ORDER BY. For example:

```
SELECT
    *
FROM
    SalesOrders
ORDER BY
    OrderDate ASC

SELECT
    *
FROM
    Contacts
ORDER BY
    (
        CASE
            WHEN Surname IS NULL THEN Title
            ELSE Surname
        END
    );
```

- To group a result set into summary rows, use GROUP BY. For example:

```
SELECT
    COUNT(Id) As "Number",
    ProductID
FROM
    SalesOrderItems
GROUP BY
    ProductID

SELECT
    COUNT(Id) As "Number",
    ProductID
FROM
    SalesOrderItems
GROUP BY
    ProductID
HAVING
    COUNT(Id) > 100;
```

- To do calculations based on result set values, use the SQL aggregate functions MIN(), MAX(), COUNT(), SUM(), and AVG(). For example:

```
SELECT Max(Quantity) FROM SalesOrderItems
SELECT Sum(Quantity) FROM SalesOrderItems
```

- To convert between compatible data types, use CAST. For example:

```
SELECT CAST(Quantity AS Char(100))FROM SalesOrderItems
```

- To fetch records that contain column values between a given range, use BETWEEN. For example:

```
SELECT ProductID FROM SalesOrderItems WHERE Quantity BETWEEN 10 AND 20
```

- To combine the result set of two or more SELECT statements, use UNION. For example:

```
SELECT City FROM Contacts
UNION
SELECT City FROM Customers
```

- To combine rows from two or more tables, use JOIN. For example:

```
SELECT SalesOrders.ID, Customers.Surname, SalesOrders.OrderDate
FROM SalesOrders
INNER JOIN Customers ON SalesOrders.CustomerID=Customers.ID;
```

- To fetch records that contain column values matching a search pattern, use LIKE. For example:

```
SELECT Surname, GivenName FROM Customers WHERE CompanyName LIKE 'R%'
SELECT Surname, GivenName FROM Customers WHERE CompanyName LIKE '_he'
```

- To search for columns without a value (NULL) or with a value (non NULL), use either IS NULL or IS NOT NULL. For example:

```
SELECT * FROM Customers WHERE CompanyName IS NULL
```

- To specify multiple values in a WHERE clause, you can use IN as an alternative to OR. For example:

```
SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region = 'Eastern'
    OR Region = 'Western'
    OR Region = 'Central'
```

can be replaced with:

```
SELECT
    OrderDate,
    SalesRepresentative
FROM
    SalesOrders
WHERE
    Region IN ('Eastern', 'Western', 'Central')
```

- To set the maximum number of records to return, use LIMIT. For example:

```
SELECT * FROM Customers LIMIT 10
```

- To test for the existence of records in a subquery, use EXISTS. For example:

```
SELECT
    Name
FROM
    Products
WHERE
    EXISTS (
        SELECT
            *
        FROM
            SalesOrderItems
        WHERE
            Products.ID = SalesOrderItems.ProductID
            AND Quantity < 20
    )
```

## Example inserts, updates, and deletes

- To insert a SQL Server record, use INSERT INTO. For example:

```
INSERT INTO
    Customers (
        Surname,
        GivenName,
        City,
        Phone,
        CompanyName
    )
VALUES
    (
        'Devlin',
        'Michaels',
        'Kingston',
        '2015558966',
        'PowerGroup'
    )
```

- Here's a SQL Server linked server example:

```
EXEC ('INSERT INTO Customers (Surname, GivenName, City, Phone, CompanyName)
VALUES ('Devlin' , 'Michaels' , 'Kingston' , '2015558966' ,
'PowerGroup'))
```

- Here's an Oracle linked table example:

```
DECLARE
    num_rows integer;
BEGIN
    num_rows:=DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@MSSQLLink
    ('INSERT INTO Customers (Surname, GivenName, City, Phone, CompanyName) VALUES
    ('Devlin', 'Michaels', 'Kingston', '2015558966', 'PowerGroup')');
END;
/
```

- The Easysoft ODBC-SQL Server Driver also supports parameterized inserts. Here's an example of doing this in [Perl](#):

```
my $sth = $dbh->prepare(q/INSERT INTO Customers (Surname, GivenName, City, Phone,
CompanyName) VALUES (?, ?, ?, ?, ?)/)
    or die "Can't prepare statement: $DBI::errstr";

$sth->execute('Devlin', 'Michaels', 'Kingston', '2015558966', 'PowerGroup');
```

- To update a SQL Server record, use UPDATE. For example:

```
UPDATE Customers
SET
    Surname = 'Jones'
WHERE
    Account_Id = 'PowerGroup'
```

The Easysoft ODBC-SQL Server Driver also supports parameterized updates. Here's an example of doing this in [Perl](#):

```
my $sth = $dbh->prepare('UPDATE Customers SET Surname = \'Jones\' WHERE
CompanyName = ?')
    or die "Can't prepare statement: $DBI::errstr";

$sth->execute('PowerGroup');
```



- To delete a SQL Server record, use DELETE. For example:

```
-- Delete (mark inactive) a bank account  
DELETE FROM Customers WHERE CompanyName = 'PowerGroup'
```

The Easysoft ODBC-SQL Server Driver also supports parameterized deletes. Here's an example of doing this in [Python](#):

```
sql = "DELETE FROM Customers WHERE CompanyName = ?"  
cursor.execute(sql, 'PowerGroup')
```

# Index

@

{PRODUCT

authentication}, 137

A

Access

importing SQL Server data, 55

linking SQL Server data, 55

Allow\_C\_Comment attribute, 43

ApplicationIntent attribute, 32

Appname attribute, 30

Authentication attribute, 26

authentication modes, 137

B

Base

working with SQL Server data, 62

bulk copy, 93

C

Client\_CSet attribute, 38

ClientLB attribute, 32

ColumnEncryption attribute, 43

connecting to SQL Server, 20

connection

troubleshooting, 46

connection failover, 150

connection string attributes, 49

ConnectionTimeout attribute, 41

ConnectRetryCount attribute, 32

ConnectRetryInterval attribute, 32

ConvToUtf attribute, 35, 37

ConvWToUtf attribute, 37

cursor support, 79

D

data source attributes, 22

data sources

removing

on Windows, 19

Database attribute, 27

database mirroring, 146

Description attribute, 23

DG4ODBC

working with SQL Server data, 58

DisguiseGuid attribute, 33

DisguiseLong attribute, 33

DPrec attribute, 33

DSN-less connections, 49

E

Easysoft ODBC-SQL Server Driver

adding ODBC data sources

on Linux or UNIX, 20

on Windows, 22

bulk copy, 93

connecting to SQL Server, 20

connection failover, 150

cursor support, 79

data source attributes, 22, 142

data types, 80

database mirroring, 146

DSN-less connections, 49

encryption, 138

installing

on Linux or UNIX, 6

on Windows, 17

IPv6 and, 151

logging, 50

ODBC API support, 76

SQL examples, 153

troubleshooting the connection, 46

Unicode support, 87

uninstalling

on Linux or UNIX, 16

on Windows, 19

encryption, 138

Excel

importing SQL Server data with Data

Connection Wizard, 56

importing SQL Server data with PowerPivot,  
56

importing SQL Server data with Query, 56

F

Failover\_Partner attribute, 32

FailoverSeverSPN attribute, 45

ForceShiloh attribute, 31

FPrec attribute, 34

G

Go

working with SQL Server data, 63

GSSFlag attribute, 45

GSSHost attribute, 45

GSSLib attribute, 45

I

installing the Easysoft ODBC-SQL Server Driver,  
6

IPv6, 151

IPv6 attribute, 41

K

Kerberos attribute, 44

- L
- Language attribute, [29](#)
- LCID attribute, [40](#)
- LibreOffice
  - working with SQL Server data, [62](#)
- LimitLong attribute, [33](#)
- log files, [50](#)
- LogFile attribute, [30](#)
- Logging attribute, [30](#)
- LogonTimeout attribute, [42](#)
- M
- MARS\_Connection attribute, [30](#)
- MAX data types, [91](#)
- MultiSubnetFailover attribute, [32](#)
- N
- Node.js
  - working with SQL Server data, [64](#)
- NTLMv2 attribute, [41](#)
- O
- ODBC API function support, [76](#)
- ODBC connection string attributes, [49](#)
- ODBC data sources
  - adding
    - on Linux or UNIX, [20](#)
    - on Windows, [22](#)
  - removing
    - on Windows, [19](#)
- Oracle
  - working with SQL Server data, [58](#)
- P
- PacketSize attribute, [43](#)
- Password attribute, [26](#)
- Perl
  - working with SQL Server data, [66](#)
- PHP
  - working with SQL Server data, [69](#)
- Port attribute, [25](#)
- Power BI
  - importing SQL Server data, [57](#)
- PowerPivot
  - importing SQL Server data, [56](#)
- PreserveCursor attribute, [31](#)
- procedures
  - named parameters, [134](#)
- Python
  - working with SQL Server data, [71](#)
- Q
- QuotedId attribute, [28](#)
- R
- R
  - working with SQL Server data, [73](#)
- RcvBuffer attribute, [42](#)
- S
- Server attribute, [24](#)
- Server\_CSet attribute, [39](#)
- Server\_UCSet attribute, [40](#)
- ServerName attribute, [27](#)
- ServerSPN attribute, [44](#)
- snapshot isolation, [92](#)
- SoKeepalive attribute, [42](#)
- SQL examples, [153](#)
- SQLServerUTF attribute, [46](#)
- SQLServerUtf attribute, [37](#)
- SSL, [138](#)
- Strfsize attribute, [34](#)
- Strftime attribute, [34](#)
- T
- table-valued parameters, [130](#)
- trace files, [50](#)
- Trusted\_Connection attribute, [41](#)
- Trusted\_Domain attribute, [40](#)
- U
- Unicode, [87](#)
- uninstalling
  - on Linux or UNIX, [16](#)
  - on Windows, [19](#)
- Use\_LCID attribute, [40](#)
- User attribute, [26](#)
- User\_Domain attribute, [28](#)
- UTF8DB attribute, [37](#)
- V
- VarMaxAsLong attribute, [33](#)
- VarMaxAsVarchar attribute, [33](#)
- Version7 attribute, [31](#)
- W
- Wsid attribute, [31](#)
- X
- XAShutdown attribute, [43](#)
- XATimeout attribute, [43](#)
- XML data type, [89](#)