ODBC SQL Server Driver

What's New

Overview

System Requirements

Installation

SQL Server Data Sources

Connecting to a Data Source

SQL Server Driver Attributes (**SQLGetInfo** Return Values)

SQL Support

Datatype Support

Using ODBC Cursors

Error Messages

Selected Functions: Notes

Driver Implementation: Notes

Glossary

References

Installation

Installing the SQL Server Driver

Upgrading the Catalog Stored Procedures

SQL Server Data Sources

ODBC SQL Server Setup

Deleting a Data Source

Using SQLConfigDataSource to Change Data Sources

Connecting to a Data Source

Connecting to a SQL Server Data Source

SQLConnect

SQLDriverConnect

SQLBrowseConnect

Driver-specific Connection Options

SQL Support

SQL Grammar Limitations

Non-supported ODBC SQL Grammar

Nullability Resolution

Datatype Support

Datatype Usage

Datatype Mapping

Nullability Resolution

Using ODBC Cursors

Using ODBC Cursors

- Creating Cursors
  - SQLSetStmtOption
  - SQLSetScrollOptions
  - SQLSetConnectOption
  - SQLSetCursorName
  - SQLGetCursorName
  - SQLGetStmtOption
  - SQLGetInfo
- Retrieving Data from Cursors
  - SQLFetch
  - SQLExtendedFetch
  - SQLRowCount
- Updating Cursors
  - SQLSetPos
  - SQLRowCount
- Closing Cursors
  - SQLFreeStmt
  - SQLTransact

Selected Functions: Notes

ODBC API Function Implementation

- SQLColAttributes, SQLDescribeCol, and SQLNumResultCols
- SQLConfigDataSource
- SQLPrepare
- SQLBrowseConnect
- SQLConnect
- SQLDriverConnect
- SQLParamOptions
- SQLDescribeParam

Driver Implementation: Notes

Active *hstmt*

Cursor Library

Arithmetic Errors

Manual-commit Mode Transactions

Remote Procedure Calls

What's New

The following features of the ODBC SQL Server driver are new in version 2.5.

Full Level 2 Compliance

The SQL Server driver supports all Level 2 ODBC APIs. Support for the following APIs is new in version 2.5.

SQLParamOptions

SQLDescribeParam

SQLSetPos

SQLExtendFetch

For details on these APIs, refer to this Help file, and to the *ODBC API Reference*.

Server Cursors

The SQL Server driver automatically uses server cursors (static, keyset, and dynamic) implemented in SQL Server 6.0. This results in improved performance, cleaner cursor semantics (compared to the cursor library in earlier versions), and more efficient memory usage at the client. Server cursors are not used in the following cases:

- The user sets ODBC_CURSORS to SQL_CUR_USE_ODBC.
- The user declares the cursor to be forward-only, read-only, rowset = 1.
- The user declares the cursor to be static and updatable.

One key benefit of using server cursors is that you can have multiple active statements (several open cursors) on the same connection. For more information about using SQL Server 6.0 server cursors with ODBC functions, see Using ODBC Cursors.

Decimal and Numeric Datatypes

The driver supports full usage of decimal and numeric datatypes and provides the appropriate convert functions to and from other datatypes.

The following ODBC functions return datatype information:

- **SQLGetInfo**
- **SQLGetTypeInfo**
- **SQLColumns**
- **SQLDescribeCol**
- **SQLProcedureColumns**
- **SQLSpecialColumns**

The following ODBC functions may involve conversions between the ODBC SQL and ODBC C datatypes:

- **SQLBindCol**
- **SQLBindParameter**
- **SQLGetData**
- **SQLPutData**

Batched RPCs

The driver automatically batches RPC requests to the server when appropriate. This could drastically reduce the number of trips to the server and result in an order of magnitude performance gain in many cases. The driver batches RPC requests when the user issues a batch of canonical stored procedure invocations or uses **SQLParamOptions** in conjunction with **SQLPrepare**/**SQLExecute** or a canonical procedure invocation.

Prepare/Execute Statements

The driver creates temporary stored procedures (as opposed to regular stored procedures in earlier versions) on a SQLPrepare. These procedures are automatically deleted when the connection is abnormally terminated. Previously, stored procedures created on a Prepare were orphaned at the server when the connection was abnormally terminated. In addition, the driver provides users with a driver specific connection option (SQL_USE_PROCEDURE_FOR_PREPARE) that governs when these temporary stored procedures are deleted. By default, they are deleted when the connection is freed. When you are issuing several prepare/execute statements in a connection and need to conserve usage of TempDB, you can have the driver delete these procedures when an *hstmt* is closed.

Use of SQL Server 6.0 Features

The driver takes advantage of the following new features in SQL Server v6.0.

- The driver uses the Set Transaction Isolation Level statement to
  - expose the Read Uncommitted isolation level
  - enforce strict serializability.

  In earlier versions, the driver "manually" inserted a HoldLock in the FROM clause of SELECT statements when the user had requested the Serializable option.
- The driver uses the SET FMTONLY statement to implement the **SQLDescribeCol** and **SQLColAttributes** APIs. Previously, these APIs were implemented by issuing a "dummy" query that did not return any results.
- The driver enforces the use of ANSI Quoted Identifiers and issues the SET QUOTED_IDENTIFIER ON statement during the start-up sequence.
- The driver exposes SQL Server's ANSI '89 IEF using the **SQLPrimaryKeys** and **SQLForeignKeys** APIs.
- The driver uses the Set ANSI_NULL_DFLT_ON statement to enforce ANSI compliant default nullability for columns. In earlier versions, the driver "manually" inserted the Null keyword after column specifications.
- The driver allows text/image parameters in stored procedures. Invocation of these stored procedures is handled as an RPC request as opposed to a language event as in previous versions.
- The driver automatically uses the default packet size set at the server if one is not specified using the ODBC **SQLSetConnectOption**.

Identity Attribute

The driver exposes the SQL Server 6.0 Identity Attribute using the SQL_COLUMN_AUTO_INCREMENT descriptor in **SQLColAttributes** and the AUTO_INCREMENT column in the **SQLGetTypeInfo** result set. SQL Server treats identity as an attribute whereas ODBC treats it as a datatype. To resolve this mis-match, **SQLGetTypeInfo** returns the following five new datatypes:

- int identity
- smallint identity

- tinyint identity
- decimal identity
- numeric identity

Text/Image Processing

The driver has several enhancements to the processing of text and image data.

- When connected to SQL Server 6.0, the driver uses the new UPDATETEXT function in the server to support updating the text/image columns of all rows when the UPDATE statement affects multiple rows. When connected to SQL Server 4.2x, only the text/image columns of the first row (of a multi-row UPDATE) will be updated.
- The driver exposes a driver specific statement option (SQL_TEXTPTR_LOGGING) to disable the logging of text/image operations.
- When using server cursors, the driver does not retrieve text/image data for unbound columns unless the user explicitly issues the **SQLGetData** call.
- When connected to SQL Server 6.0, **SQLExecDirect** and **SQLExecute** do not require the SQL_LEN_DATA_AT_EXEC macro, but using the macro is more efficient.

Performance Improvements

The driver has added the following performance-related features:

- Improved Connect TimeThe driver batches informational queries to the server when making a connection that reduces the number of trips to the server. The fast connect option is no longer needed or supported.
- The driver makes better use of memory (eliminating copies) for faster processing of results.
- 64-bit ArithmeticThe 32-bit driver uses 64-bit arithmetic for data conversions to/from *money*, *decimal* and *numeric* datatypes.

Integrated Security

The driver provides a driver specific option (SQL_INTEGRATED_SECURITY) to request a secure connection from SQL Server 6.0.

SQLExtendedFetch for 4.2a SQL Server

The driver supports the **SQLExtendedFetch** API (forward-only, read-only cursors) for SQL Server 4.21a.

DSN

When you install the 32-bit driver, you have the option to convert existing 16-bit DSNs to 32-bit DSNs so they can be used by 32-bit applications. In addition, the driver uses the OEMtoANSI setting from the Client Configuration Utility as the default when a new DSN is defined.

DayOfWeek

The driver supports the DayOfWeek canonical function defined in ODBC.

Literal Params in Procedures

The driver executes canonical procedure invocations that have literal parameters as an RPC request. In earlier versions, unless the invocation had all parameter markers, the request was issued as a

language event.

SQLGetInfo Changes

For more information, see the **SQLGetInfo** function.

INSTCAT.SQL

The catalog stored procedures expose the new functionality added in version 2.5 and improves performance over previous versions.

DBCS

The version 2.5 driver is DBCS enabled.

Arithmetic Errors

The version 2.5 driver uses SET ARITHABORT ON to provide better ODBC semantics.

ODBC SDK

The ODBC SDK is included as part of the Microsoft SQL Workstation version 6.0.

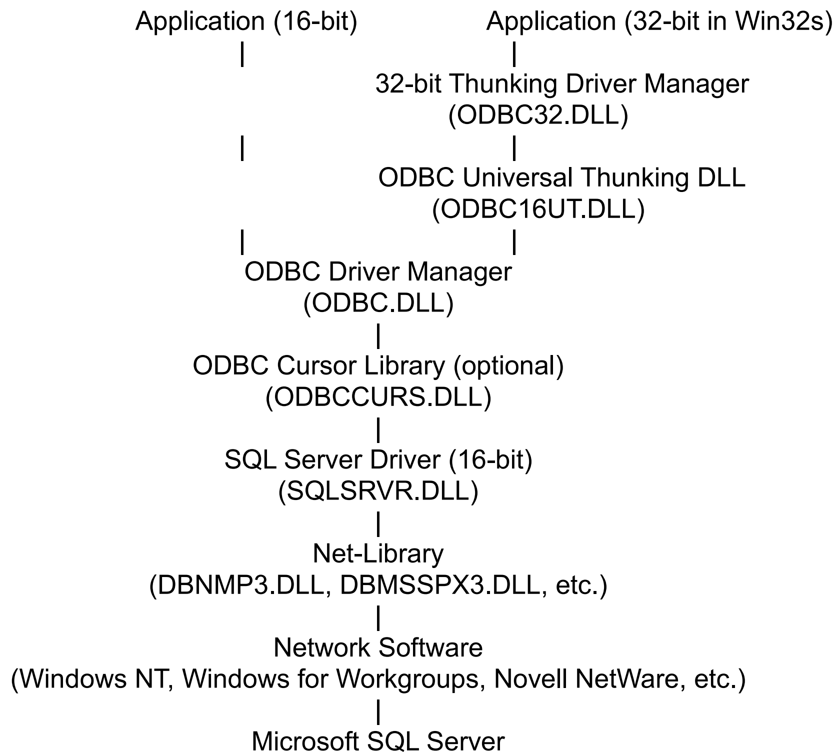
Overview

Microsoft SQL Server is a multiuser relational database management system (DBMS) that runs on the Microsoft Windows NT operating system. Both 16- and 32-bit versions of the SQL Server ODBC driver are available. Structured Query Language (SQL) is used to access data in a SQL Server database. Client workstations communicate with SQL Server across a network, such as a Windows NT Server, Novell NetWare, Banyan VINES, or TCP/IP network.

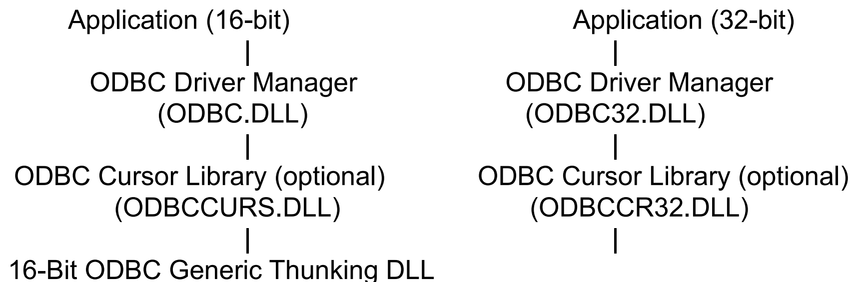
The SQL Server driver enables applications to access data in Microsoft SQL Server databases through the Open Database Connectivity (ODBC) interface.

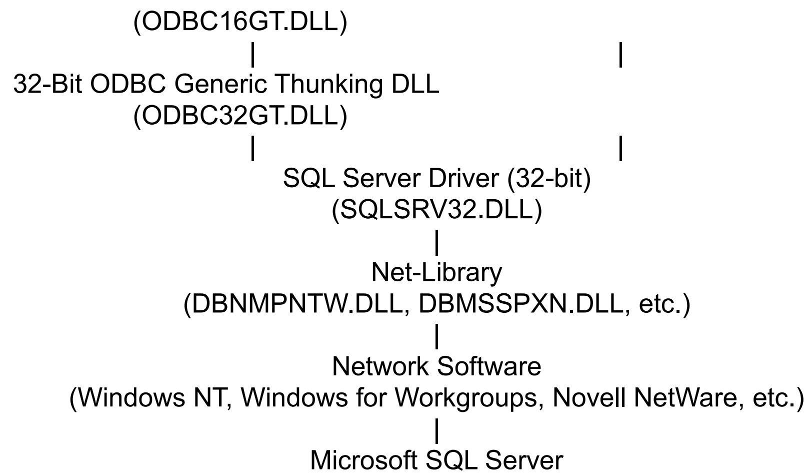
Note that the 32-bit SQL Server driver is thread safe. The driver will serialize shared access by multiple threads to shared *hstmt*, *hdbc*, and *henv* objects. However, the ODBC program is still responsible for keeping operations within statement and connection spaces in the proper sequence, even when the program uses multiple threads.

This is the application/driver architecture for 16-bit environments:



This is the application/driver architecture for 32-bit environments:





Thunking means intercepting a function call, doing some special processing to translate between 16-bit and 32-bit code, and then transferring control to a target function.

See Also

Installing the SQL Server Driver

ODBC SQL Server Setup

Connecting to a SQL Server Data Source

System Requirements

System Requirements

To access SQL Server data, you must have:

- The SQL Server driver.
- Microsoft SQL Server.
- Network software that the computers on which the driver and SQL Server reside (not required when connecting to a local (non-network) desktop SQL Server).

The hardware and software requirements of each of these components follows.

SQL Server Driver

Hardware Requirements:

- An Industry Standard Architecture (ISA) computer, such as the IBM PC/AT or compatible,

  Or

  A Micro Channel Architecture (MCA) computer, such as an IBM PS/2 or compatible,

  Or

  An Extended Industry Standard Architecture (EISA) computer with an 80286, 80386, 80486, or Pentium microprocessor,

  Or

  An Alpha AXP or MIPS microprocessor.
- At least 2 MB of RAM; 4 MB recommended.
- A hard disk drive with approximately 300K of hard disk space for the SQL Server driver and ODBC Driver Manager.

Software Requirements:

- Microsoft Windows version 3.1 or later (16-bit driver)
- Microsoft Windows 95 or Windows NT 3.5 or later (32-bit driver)

For information about the hardware and software required for SQL Server clients, see *Microsoft SQL Server Setup*.

SQL Server

To use the SQL Server driver to access data in Microsoft SQL Server databases, you must have Microsoft SQL Server version 4.21A or later. The catalog stored procedures must be installed on your SQL Server. You may need to install the catalog stored procedures shipped with this driver when using version 4.21A of Microsoft SQL Server (see the Upgrading the Catalog Stored Procedures topic). For information about the hardware and software required by SQL Server, see *Microsoft SQL Server Setup*.

Network Software

Network software is required to connect the computers on which the SQL Server driver and SQL Server reside. To connect to a SQL Server, you can use Microsoft Windows NT, Microsoft Windows for Workgroups, or a compatible network such as IBM LAN Server or DEC Pathworks, Novell NetWare, or Banyan VINES. For information about the hardware and software required by each network, see your network documentation.

The SQL Server driver communicates with network software through the SQL Server NetLibrary interface, which requires a Net-Library dynamic-link library (DLL). For more information about

supported network configurations and Net-Library files, see *Microsoft SQL Server Setup*.

See Also

Installing the SQL Server Driver

Installing the SQL Server Driver

The SQL Server ODBC driver is installed automatically when you install the SQL Server 6.0 client software on a computer running Windows NT, Windows 95, or Windows 3.11. For more information about installing SQL Server 6.0 client software, see *Microsoft SQL Server Setup*.

If you need to reinstall just the SQL Server ODBC driver, you can run the ODBC **setup** program from the ODBC subdirectory of the appropriate client install directory.

To use the SQL Server driver you usually add a data source for each installation of SQL Server in which you want to access data. See the SQL Server Data Sources topic.

If you are using Microsoft SQL Server version 4.21A, the INSTCAT.SQL included with this driver includes minor fixes to the catalog stored procedures. To install the updated catalog stored procedures, follow the instructions for Upgrading the Catalog Stored Procedures.

See Also

SQL Server Data Sources

System Requirements

Error Messages

When an error occurs, the SQL Server driver returns the native error number, the SQLSTATE (an ODBC error code), and an error message. The driver derives this information both from errors detected by the driver and errors returned by SQL Server.

Native Error Numbers

For errors that occur in the data source (returned by SQL Server), the SQL Server driver returns the native error number returned to it by SQL Server. For errors detected by the driver, the SQL Server driver returns a native error number of zero. For a list of native error numbers, see the error column of the *sysmessages* system table in the *master* database in SQL Server.

SQLSTATE (ODBC Error Codes)

For errors that occur in the data source that are detected and returned by SQL Server, the SQL Server driver maps the returned native error number to the appropriate SQLSTATE. If a native error number does not have an ODBC error code to map to, the SQL Server driver returns SQLSTATE 37000 (Syntax error or access violation). For errors that are detected by the driver, the SQL Server driver generates the appropriate SQLSTATE.

Error Messages

For errors that occur in the data source and are detected and returned by SQL Server, the SQL Server driver returns an error message based on the message returned by SQL Server. For errors that occur in the SQL Server driver or the Driver Manager, the SQL Server driver returns an error message based on the text associated with the SQLSTATE. For a list of error messages that can be returned by SQL Server (but not by the SQL Server driver), see the description column of the *sysmessages* system table in the *master* database in SQL Server.

Syntax

Error messages have the following format:

[vendor][ODBC_component][data_source]error_message

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

Note When the error occurs in the data source, the *[vendor]* and *[ODBC_component]* prefixes identify the vendor and name of the ODBC component that received the error from the data source.

Data source	Prefix	Value
Driver Manager	[*vendor*] [*ODBC_component*] [*data_source*]	[Microsoft] [ODBC Driver Manager] N/A
Cursor Library	[*vendor*] [*ODBC_component*] [*data_source*]	[Microsoft] [ODBC Cursor Library] N/A
SQL Server driver	[*vendor*] [*ODBC_component*] [*data_source*]	[Microsoft] [ODBC SQL Server Driver] N/A
SQL Server	[*vendor*] [*ODBC_component*]	[Microsoft] [ODBC SQL Server

[*data_source*]	Driver] [SQL Server]

Upgrading the Catalog Stored Procedures

The SQL Server driver uses a set of system stored procedures, known as catalog stored procedures, to obtain information from the SQL Server system catalog. Microsoft SQL Server installs the catalog stored procedures automatically when you install or upgrade SQL Server. The INSTCAT.SQL file included with this driver includes minor updates to the catalog stored procedures. Having the SQL Server system administrator upgrade the catalog stored procedures on a server running Microsoft SQL Server version 4.21A is recommended but not required.

To upgrade the catalog stored procedures, the system administrator runs a script using the **isql** utility (see the instructions below). Before making any changes to the *master* database, the system administrator should back it up. To run **isql**, your computer must be installed as a client workstation for Microsoft SQL Server.

To upgrade the catalog stored procedures

- At the command prompt, use the **isql** utility to run the INSTCAT.SQL script. For example, C:> **ISQL /Usa /P***sa_password* **/S***server_name* **/i***location*\INSTCAT.SQL

  where

  location is the full path of the location of INSTCAT.SQL. You can use the INSTCAT.SQL from an installed SQL Server 6.0 (the default location is C:\SQL60\INSTALL) or from the SQL Server 6.0 CD (the default location is *D*:*platform* where *D* is the CD drive letter and *platform* is the appropriate server platform directory, such as I386).

  sa_password is the system administrator's password.

  server_name is the name of the server on which SQL Server resides.

The INSTCAT.SQL script generates messages. Ignore these.

The INSTCAT.SQL script fails when there is not enough room in the *master* database to store the catalog stored procedures or to log the changes to existing procedures. If the INSTCAT.SQL script fails, contact your system administrator.

The SQL Server driver uses the following catalog stored procedures.

Stored Procedure	Returns
sp_column_privileges	Information about column privileges for the specified table(s)
sp_columns	Information about columns for the specified table(s)
sp_databases	A list of databases
sp_datatype_info	Information about the supported datatypes
sp_fkeys	Information about logical foreign keys
sp_pkeys	Information about primary keys
sp_server_info	A list of attribute names and matching values for the server
sp_special_columns	Information for a single table about columns in the table that have special attributes
sp_sproc_columns	Column information for a stored procedure
sp_statistics	A list of indexes for a single table
sp_stored_procedures	A list of stored procedures
sp_table_privileges	Information about table privileges for the

	specified table(s)
sp_tables	A list of objects that can be queried

For additional information about the catalog stored procedures, see the *Microsoft SQL Server Transact-SQL Reference*.

ODBC SQL Server Setup

To access the ODBC SQL Server Setup dialog box

1. From Control Panel, double-click the ODBC icon.

   Note When using 16-bit drivers on Microsoft Windows NT, in the Program Manager window, open the Microsoft ODBC group and double-click the Microsoft ODBC Administrator icon.

2. The Data Sources dialog box appears. Choose the Add button.

   The Add Data Source dialog box appears.

3. From the Installed ODBC Drivers list, select SQL Server, and then choose OK.

   The ODBC SQL Server Setup dialog box appears.

The ODBC SQL Server Setup dialog box has the following options.

Data Source Name

The name of the data source. For example, "Personnel Data."

Description

A description of the data in the data source. For example, "Hire date, salary history, and current review of all employees."

Server

The name of a SQL Server on your network. You can select a server from the list or enter the server name.

"(local)" can be entered as the server on a Microsoft Windows NT computer. The user can then use a local copy of SQL Server (that listens on named pipes), even when running a non-networked version of SQL Server. Note that when the 16-bit SQL Server driver is using "(local)" without a network, the MS Loopback Adapter must be installed.

For more information about server names for different types of networks, see *Microsoft SQL Server Setup*.

Network Address

The address of the SQL Server database management system (DBMS) from which the driver retrieves data. For Microsoft SQL Server you can usually leave this value set to (Default).

Network Library

The name of the SQL Server Net-Library DLL that the SQL Server driver uses to communicate with the network software. If the value of this option is (Default) the SQL Server driver uses the client computer's default Net-Library, which is specified in the Default Network box in the Net-Library tab of the SQL Server Client Configuration Utility.

If you create a data source using a Network Library other than (Default) and optionally a Network Address, ODBC SQL Server Setup will create a server name entry that you can see in the Advanced tab in the SQL Server Client Configuration Utility. These server name entries can also be used by DB-Library applications.

Options

To access the following fields, click the Options button.

Database Name

The name of the SQL Server database.

Language Name

The national language used by SQL Server.

Generate Stored Procedures for Prepared Statements

Stored procedures are created for prepared statements when this option is selected (the default). The SQL Server driver prepares a statement by placing it in a procedure and compiling that procedure.

When this option checkbox is clear, the creation of stored procedures for prepared statements is disabled. In this case, a prepared statement is stored and executed at execution time.

Translation

The description of the current translator. To select a different translator, choose the Select button and select from the list in the Select Translator dialog box.

Convert OEM to ANSI Characters

If the SQL Server client computer and SQL Server are using the same non-ANSI character set select this option. For example, if SQL Server uses code page 850 and this client computer uses code page 850 for the OEM code page, selecting this option will ensure that extended characters stored in the database are property converted to ANSI for use by Windows-based applications.

When this option checkbox is clear and the SQL Server client machine and SQL Server are using different character sets, you must specify a character set translator.

See Also

Driver-specific Connection Options

Using SQLConfigDataSource to Change Data Sources

Using SQLConfigDataSource to Change Data Sources

You can call the SQLConfigDataSource API function to add, modify, or delete a data source dynamically. This function uses keywords to set connection options that are normally set through the ODBC SQL Server Setup dialog box. Use this function when you want to add, modify, or delete a data source without displaying the ODBC SQL Server Setup dialog box.

Deleting a Data Source

To delete a data source, in the Data Sources dialog box select the data source you want to delete from the Data Sources list. Choose the Delete button, and then choose the Yes button to confirm the deletion.

You may be asked if you want to remove the data source name and its associated information from the WIN.INI file or registry. Other applications that call SQL Server may use this information to connect to SQL Server data sources.

Choose the Yes button if you are certain that no other applications use the information about the data source; otherwise, choose the No button.

SQLGetInfo Return Values

The following table lists the C language #defines for the *fInfoType* argument and the corresponding values returned by **SQLGetInfo** when connected to SQL Server 6.0. An application retrieves this information by passing the listed C language #defines to **SQLGetInfo** in the *fInfoType* argument.

***fInfoType* argument (#define)**	**Value returned by SQLGetInfo**
SQL_ACCESSIBLE_PROCEDURES	Y
SQL_ACCESSIBLE_TABLES	Y
SQL_ACTIVE_CONNECTIONS	0 (The number of active connections is determined by the number of network connections available on the client computer and the number of connections allowed by the server DBMS.)
SQL_ACTIVE_STATEMENTS	1 (Even though this value is 1, the driver supports multiple active statements on a connection if server cursors are being used. See Using ODBC Cursors).
SQL_ALTER_TABLE	SQL_AT_ADD_COLUMN
SQL_BOOKMARK_PERSISTENCE	0
SQL_COLUMN_ALIAS	Y
SQL_CONCAT_NULL_BEHAVIOR	SQL_CB_NON_NULL
SQL_CONVERT_FUNCTIONS	SQL_FN_CVT_CONVERT
SQL_CONVERT_*type* where *type* is the SQL datatype, such as CHAR	See the Datatype Mapping topic.
SQL_CORRELATION_NAME	SQL_CN_ANY
SQL_CURSOR_COMMIT_BEHAVIOR	SQL_CB_CLOSE (This is the ANSI-specified behavior. Recognizing the need for many applications to preserve server cursor currency across commits/rollbacks, the driver provides a driver-specific option, SQL_PRESERVE_CURSORS, to override the ANSI default.)
SQL_CURSOR_ROLLBACK_BEHAVIOR	SQL_CB_CLOSE (This is the ANSI-specified behavior. Recognizing the need for many applications to preserve server cursor currency across commits/rollbacks, the

	driver provides a driver-specific option, SQL_PRESERVE_CURSORS, to override the ANSI default.)
SQL_DBMS_NAME	Microsoft SQL Server
SQL_DEFAULT_TXN_ISOLATIO N	SQL_TXN_READ_COMMITTED .
SQL_DRIVER_NAME	SQLSRVR.DLL (16-bit) SQLSRV32.DLL (32-bit)
SQL_DRIVER_ODBC_VER	02.50
SQL_DRIVER_VER	02.50.*nnnn* (where *nnnn* specifies the build number)
SQL_EXPRESSIONS_IN_ORDE RBY	Y
SQL_FETCH_DIRECTION	SQL_FD_FETCH_NEXT SQL_FD_FETCH_FIRST SQL_FD_FETCH_LAST SQL_FD_FETCH_PRIOR SQL_FD_FETCH_ABSOLUTE SQL_FD_FETCH_RELATIVE
SQL_FILE_USAGE	SQL_FILE_NOT_SUPPORTED
SQL_GETDATA_EXTENSIONS	SQL_GD_BLOCK
SQL_GROUP_BY	SQL_GB_GROUP_BY_CONTAI NS_SELECT
SQL_IDENTIFIER_CASE	(Depends on whether SQL Server was installed as case-sensitive or not case-sensitive.)
SQL_IDENTIFIER_QUOTE_CHA R	" (Double quote)
SQL_KEYWORDS	BREAK BROWSE BULK CHECKPOINT CLUSTERED COMMITTED COMPUTE CONFIRM CONTROLROW DATABASE DBCC DISK DUMMY DUMP ERRLVL ERROREXIT EXIT EXPIREDATE FILE FILLFACTOR FLOPPY

	GETDEFAULT HOLDLOCK IDENTITY_INSERT IDENTITYCOL IF KILL LINENO LOAD MIRROREXIT NONCLUSTERED OFF OFFSETS ONCE OVER PERM PERMANENT PLAN PRINT PROC PROCESSEXIT PUBLIC RAISERROR READ READTEXT RECONFIGURE REPEATABLE RETAINDATE RETURN ROWCOUNT RULE SAVE SERIALIZABLE SETUSER SHUTDOWN STATISTICS TAPE TEMP TEXTSIZE TRAN TRIGGER TRUNCATE TSEQUEL UNCOMMITTED UPDATETEXT USE VOLUME WAITFOR WHILE WRITETEXT
SQL_LIKE_ESCAPE_CLAUSE	N
SQL_LOCK_TYPES	SQL_LCK_NO_CHANGE
SQL_MAX_BINARY_LITERAL_LEN	131072
SQL_MAX_CHAR_LITERAL_LE	131072

N	
SQL_MAX_COLUMN_NAME_LEN	30
SQL_MAX_COLUMNS_IN_GROUP_BY	16
SQL_MAX_COLUMNS_IN_INDEX	16
SQL_MAX_COLUMNS_IN_ORDER_BY	16
SQL_MAX_COLUMNS_IN_SELECT	4000
SQL_MAX_COLUMNS_IN_TABLE	250
SQL_MAX_CURSOR_NAME_LEN	30
SQL_MAX_INDEX_SIZE	256
SQL_MAX_OWNER_NAME_LEN	30
SQL_MAX_PROCEDURE_NAME_LEN	36 (1 to 30 characters followed by a semicolon [;] and one to five digits)
SQL_MAX_QUALIFIER_NAME_LEN	30
SQL_MAX_ROW_SIZE	1962
SQL_MAX_ROW_SIZE_INCLUDES_LONG	N
SQL_MAX_STATEMENT_LEN	131072
SQL_MAX_TABLE_NAME_LEN	30
SQL_MAX_TABLES_IN_SELECT	16
SQL_MAX_USER_NAME_LEN	30
SQL_MULT_RESULT_SETS	Y
SQL_MULTIPLE_ACTIVE_TXN	Y
SQL_NEED_LONG_DATA_LEN	Y
SQL_NON_NULLABLE_COLUMNS	SQL_NNC_NON_NULL
SQL_NULL_COLLATION	SQL_NC_LOW
SQL_NUMERIC_FUNCTIONS	SQL_FN_NUM_ABS SQL_FN_NUM_ACOS SQL_FN_NUM_ASIN SQL_FN_NUM_ATAN SQL_FN_NUM_ATAN2 SQL_FN_NUM_CEILING SQL_FN_NUM_COS SQL_FN_NUM_COT

	SQL_FN_NUM_DEGREES SQL_FN_NUM_EXP SQL_FN_NUM_FLOOR SQL_FN_NUM_LOG SQL_FN_NUM_LOG10 SQL_FN_NUM_MOD SQL_FN_NUM_PI SQL_FN_NUM_POWER SQL_FN_NUM_RADIANS SQL_FN_NUM_RAND SQL_FN_NUM_ROUND SQL_FN_NUM_SIGN SQL_FN_NUM_SIN SQL_FN_NUM_SQRT SQL_FN_NUM_TAN
SQL_ODBC_API_CONFORMANCE	SQL_OAC_LEVEL2
SQL_ODBC_SAG_CLI_ CONFORMANCE	SQL_OSCC_NOT_COMPLIANT
SQL_ODBC_SQL_CONFORMANCE	SQL_OSC_CORE
SQL_ODBC_SQL_OPT_IEF	Y
SQL_ORDER_BY_COLUMNS_IN_	N
SELECT_SQL_OJ_CAPABILITIES	SQL_OJ_LEFT SQL_OJ_RIGHT SQL_OJ_NESTED
SQL_OUTER_JOINS	Y
SQL_OWNER_TERM	owner
SQL_OWNER_USAGE	SQL_OU_DML_STATEMENTS SQL_OU_PROCEDURE_ INVOCATION SQL_OU_TABLE_DEFINITION SQL_OU_INDEX_DEFINITION SQL_OU_PRIVILEGE_ DEFINITION
SQL_POS_OPERATIONS	SQL_POS_ADD SQL_POS_DELETE SQL_POS_POSITION SQL_POS_REFRESH SQL_POS_UPDATE
SQL_POSITIONED_STATEMENTS	SQL_PS_POSITIONED_DELETE SQL_PS_POSITIONED_UPDATE SQL_PS_SELECT_FOR_ UPDATE
SQL_PROCEDURE_TERM	stored procedure
SQL_PROCEDURES	Y
SQL_QUALIFIER_LOCATION	SQL_QL_START
SQL_QUALIFIER_NAME_	.

SEPARATOR	(period)
SQL_QUALIFIER_TERM	database
SQL_QUALIFIER_USAGE	SQL_QU_DML_STATEMENTS, SQL_QU_PROCEDURE_ INVOCATION, SQL_QU_TABLE_DEFINITION
SQL_QUOTED_IDENTIFIER_CA SE	(Depends on whether SQL Server was installed as case-sensitive or not case- sensitive.)
SQL_ROW_UPDATES	N
SQL_SCROLL_CONCURRENC Y	SQL_SCCO_LOCK SQL_SCCO_OPT_ROWVER SQL_SCCO_OPT_VALUES SQL_SCCO_READ_ONLY
SQL_SCROLL_OPTIONS	SQL_SO_DYNAMIC SQL_SO_FORWARD_ONLY SQL_SO_KEYSET_DRIVEN SQL_SO_STATIC
SQL_SEARCH_PATTERN_ESC APE	"\" (backslash)
SQL_SPECIAL_CHARACTERS	#$ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔ ÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëì íîïðñòóôõöøùúûüýþÿ
SQL_STATIC_SENSITIVITY	SQL_SS_ADDITIONS SQL_SS_UPDATES
SQL_STRING_FUNCTIONS	SQL_FN_STR_ASCII, SQL_FN_STR_CHAR, SQL_FN_STR_CONCAT, SQL_FN_STR_DIFFERENCE, SQL_FN_STR_INSERT, SQL_FN_STR_LCASE, SQL_FN_STR_LEFT, SQL_FN_STR_LENGTH, SQL_FN_STR_LOCATE_2, SQL_FN_STR_LTRIM, SQL_FN_STR_REPEAT, SQL_FN_STR_RIGHT, SQL_FN_STR_RTRIM, SQL_FN_STR_SOUNDEX, SQL_FN_STR_SPACE, SQL_FN_STR_SUBSTRING, SQL_FN_STR_UCASE
SQL_SUBQUERIES	SQL_SQ_COMPARISON, SQL_SQ_EXISTS, SQL_SQ_IN, SQL_SQ_QUANTIFIED, SQL_SQ_CORRELATED_ SUBQUERIES
SQL_SYSTEM_FUNCTIONS	SQL_FN_SYS_DBNAME, SQL_FN_SYS_IFNULL, SQL_FN_SYS_USERNAME,

SQL_TABLE_TERM	table
SQL_TIMEDATE_ADD_INTERVALS	SQL_FN_TSI_FRAC_SECOND, SQL_FN_TSI_SECOND, SQL_FN_TSI_MINUTE, SQL_FN_TSI_HOUR, SQL_FN_TSI_DAY, SQL_FN_TSI_WEEK, SQL_FN_TSI_MONTH, SQL_FN_TSI_QUARTER, SQL_FN_TSI_YEAR
SQL_TIMEDATE_DIFF_INTERVALS	SQL_FN_TSI_FRAC_SECOND, SQL_FN_TSI_SECOND, SQL_FN_TSI_MINUTE, SQL_FN_TSI_HOUR, SQL_FN_TSI_DAY, SQL_FN_TSI_WEEK, SQL_FN_TSI_MONTH, SQL_FN_TSI_QUARTER, SQL_FN_TSI_YEAR,
SQL_TIMEDATE_FUNCTIONS	SQL_FN_TD_NOW, SQL_FN_TD_CURDATE, SQL_FN_TD_DAYOFMONTH, SQL_FN_TD_DAYOFWEEK, SQL_FN_TD_DAYOFYEAR, SQL_FN_TD_DAYNAME, SQL_FN_TD_MONTH, SQL_FN_TD_MONTHNAME, SQL_FN_TD_QUARTER, SQL_FN_TD_WEEK, SQL_FN_TD_YEAR, SQL_FN_TD_CURTIME, SQL_FN_TD_HOUR, SQL_FN_TD_MINUTE, SQL_FN_TD_SECOND, SQL_FN_TD_TIMESTAMPADD, SQL_FN_TD_TIMESTAMPDIFF
SQL_TXN_CAPABLE	SQL_TC_DML
SQL_TXN_ISOLATION_OPTION	SQL_TXN_READ_COMMITTED SQL_TXN_READ_UNCOMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE
SQL_UNION	SQL_U_UNION, SQL_U_UNION_ALL

Convert From \ Convert To	SQL_CHAR	SQL_VARCHAR	SQL_LONGVARCHAR	SQL_DECIMAL	SQL_NUMERIC	SQL_BIT	SQL_TINYINT	SQL_SMALLINT	SQL_INTEGER	SQL_BIGINT	SQL_REAL	SQL_FLOAT	SQL_DOUBLE	SQL_BINARY	SQL_VARBINARY	SQL_LONGVARBINARY	SQL_DATE	SQL_TIME	SQL_TIMESTAMP
SQL_CHAR	•	•	•	•		•	•	•	•		•	•		•	•	•			•
SQL_VARCHAR	•	•	•	•		•	•	•	•		•	•		•	•	•			•
SQL_LONGVARCHAR	•	•	•																
SQL_DECIMAL	•	•		•		•	•	•	•		•	•		•	•				
SQL_NUMERIC																			
SQL_BIT	•	•				•	•	•	•		•	•		•	•				
SQL_TINYINT	•	•		•		•	•	•	•		•	•		•	•				
SQL_SMALLINT	•	•		•		•	•	•	•		•	•		•	•				
SQL_INTEGER	•	•		•		•	•	•	•		•	•		•	•				
SQL_BIGINT																			
SQL_REAL	•	•		•		•	•	•	•		•	•							
SQL_FLOAT	•	•		•		•	•	•	•		•	•							
SQL_DOUBLE																			
SQL_BINARY	•	•					•	•	•					•	•	•			
SQL_VARBINARY	•	•					•	•	•					•	•	•			
SQL_LONGVARBINARY														•	•	•			
SQL_DATE																			
SQL_TIME																			
SQL_TIMESTAMP	•	•												•	•				•

Connecting to a SQL Server Data Source

When an application connects to SQL Server, using **SQLDriverConnect**, the application may generate a prompt for data source information. This prompt can be in the form of a Data Source query or a dialog box. For a Data Source query, enter the name of the data source. For a dialog box, complete the dialog box as follows:

1. Enter or select the server name.
2. In the Login ID entry field, type your SQL Server login ID.
3. In the Password entry field, type your SQL Server password.
4. Choose OK.

For information about the SQL Server login security mode and secure connections, see the *Microsoft SQL Server Administrator's Companion*.

To enter optional connection information, such as the database to access and the language for SQL Server to use, choose the Options button.

1. From the Database list box, select the database you want to access.*
   When you open the list box, the user's default database is selected.
2. From the Language box, select the language you want SQL Server to use.*
   When you open the list box, the user's default language is selected.
3. If the application name displayed is incorrect, enter the correct application name.
   The application name is the name of the application that is calling the SQL Server driver.
4. If the workstation ID displayed is incorrect, enter the correct workstation ID.
   Typically, this is the network name of the computer on which the application resides.
5. Choose OK.

* Setting the default database and language for the login ID in SQL Server is more efficient than specifying them as DSN options.

Using ODBCPING.EXE to Verify a Connection

You can use the ODBCPING.EXE utility to check whether ODBC is properly installed by connecting to a server using the ODBC SQL Server Driver. This utility is a 32-bit application that is stored in the \SQL60\BINN directory. To verify ODBC connectivity, from a command prompt, type:

odbcping /S*servername* **/U***login_id* **/P***password*

where

servername
   Is the name of a server you will connect to.

login_id
   Is a valid login ID for that server.

password
   Is the password for that login ID.

If the ODBC connection is established, this message is displayed:

```
CONNECTED TO SQL SERVER
```

If the ODBC connection cannot be established, this message is displayed:

```
COULD NOT CONNECT TO SQL SERVER
```

See Also

ODBC SQL Server Setup

SQLBrowseConnect

SQLConnect

SQLDriverConnect

data source (SQL Server)

A data source includes the data a user wants to access and the information needed to access that data. For the SQL Server driver, a data source is a Microsoft SQL Server database, the server on which it resides, and the network used to access that server.

Driver-specific Connection Options

SQL Server driver supports the following driver-specific connection options in **SQLSetConnectOption** and **SQLGetConnectOption**.

Option	Description
SQL_INTEGRATED_SECURITY	SQL_IS_OFF. Request a normal connection to SQL Server. A trusted connection using SQL Server integrated security is not requested. (Default) SQL_IS_ON. Request a trusted connection to SQL Server regardless of the security mode of the server. For information about the SQL Server login security mode and trusted connections, see the *Microsoft SQL Server Administrator's Companion*.
SQL_USE_PROCEDURE _FOR_PREPARE	SQL_UP_ON. Temporary stored procedures are generated for **SQLPrepare**. (Default) SQL_UP_OFF. Temporary stored procedures are not generated for **SQLPrepare**. The statement is stored, compiled, and executed at execution time. All syntax error checking is delayed until execution time. SQL_UP_ON_DROP. Temporary stored procedures are explicitly dropped on a subsequent call to **SQLPrepare** or when an *hstmt* is freed.
SQL_PRESERVE_CURSORS	SQL_PC_OFF. All cursors are closed on **SQLTransact**. (Default) SQL_PC_ON. Server cursors remain open on **SQLTransact**.

Driver-specific Option #Defines

The following are SQL Server-specific defines.

```
//        SQLSetConnectOption/SQLSetStmtOption driver
//        specific defines 1200-1249 are reserved.

//        Connection Options
#define   SQL_USE_PROCEDURE_FOR_PREPARE          1202
#define   SQL_INTEGRATED_SECURITY *              1203
#define   SQL_PRESERVE_CURSORS                   1204

//        Statement Options
#define   SQL_TEXTPTR_LOGGING                    1225

//        SQL_USE_PROCEDURE_FOR_PREPARE Defines
#define   SQL_UP_OFF                             0L
#define   SQL_UP_ON                              1L
```

```
#define    SQL_UP_ON_DROP                  2L
#define    SQL_UP_DEFAULT                  SQL_UP_ON

//         SQL_INTEGRATED_SECURITY Defines
#define    SQL_IS_OFF                      0L
#define    SQL_IS_ON                       1L
#define    SQL_IS_DEFAULT                  SQL_IS_OFF

//         SQL_PRESERVE_CURSORS Defines
#define    SQL_PC_OFF                      0L
#define    SQL_PC_ON                       1L
#define    SQL_PC_DEFAULT                  SQL_PC_OFF

//         SQL_TEXTPTR_LOGGING Defines
#define    SQL_TL_OFF                      0L
#define    SQL_TL_ON                       1L
#define    SQL_TL_DEFAULT                  SQL_TL_ON
```

* Only useable before connecting.

Datatype Mapping

The SQL Server driver maps SQL Server SQL datatypes to ODBC SQL datatypes. The following table lists all SQL Server SQL datatypes and shows the ODBC SQL datatypes to which they are mapped.

SQL Server datatypes	ODBC SQL datatype
binary	SQL_BINARY
bit	SQL_BIT
char, character	SQL_CHAR
datetime	SQL_TIMESTAMP
decimal, dec (SQL Server 6.0)	SQL_DECIMAL
float, double precision, float(*n*) for *n* = 8-15	SQL_FLOAT
image	SQL_LONGVARBINARY
int, integer	SQL_INTEGER
money	SQL_DECIMAL
numeric (SQL Server 6.0)	SQL_NUMERIC
real, float(*n*) for *n* = 1-7	SQL_REAL
smalldatetime	SQL_TIMESTAMP
smallint	SQL_SMALLINT
smallmoney	SQL_DECIMAL
sysname	SQL_VARCHAR
text	SQL_LONGVARCHAR
timestamp*	SQL_BINARY (SQL Server 6.0) SQL_VARBINARY (SQL Server 4.2*x*)
tinyint	SQL_TINYINT
varbinary, binary varying	SQL_VARBINARY
varchar, character varying, char varying	SQL_VARCHAR

* The timestamp datatype is converted to the SQL_VARBINARY or SQL_BINARY datatype because values in timestamp columns are not datetime data, but varbinary(8) or binary(8) data, indicating the sequence of SQL Server activity on the row.

Note The SQL Server driver does not convert SQL data of types SQL_CHAR, SQL_VARCHAR, or SQL_LONGVARCHAR to C data of types SQL_C_DATE or SQL_C_TIME. All other conversions are supported for the ODBC SQL datatypes listed in this topic. For information on supported conversions, see Appendix D of the *Microsoft ODBC SDK Programmer's Reference*.

Datatype Usage

The SQL Server driver and SQL Server impose the following usage of datatypes.

Datatype	Limitation
Date literals	Date literals, when stored in an SQL_TIMESTAMP column (SQL Server types of datetime or smalldatetime), have a time value of 12:00:00.000 A.M. (midnight).
money and smallmoney	Only the integer parts of the *money* and *smallmoney* datatypes are significant. If the decimal part of SQL money data is truncated during datatype conversion, the SQL Server driver returns a warning, not an error.
SQL_BINARY	If an SQL_BINARY column is nullable, the data stored in the data source is not padded with zeroes. When data from such a column is retrieved, the SQL Server driver pads it with zeroes on the right. However, data created in operations performed by SQL Server, such as concatenation, doesn't have such padding.
SQL_CHAR (truncation)	When data is placed into a SQL_CHAR column, SQL Server truncates it on the right without warning if it is too long to fit into the column.
SQL_CHAR (nullable)	If a SQL_CHAR column is nullable, the data stored in the data source is not padded with blanks. When data from such a column is retrieved, the SQL Server driver pads it with blanks on the right. However, data created in operations performed by SQL Server, such as concatenation, doesn't have such padding.
SQL_LONGVAR BINARY, SQL_LONGVAR CHAR	Updates of columns with SQL_LONGVARBINARY or SQL_LONGVARCHAR datatypes (using a WHERE clause) that affect multiple rows are fully supported when connected to SQL Server 6.0. When connected to SQL Server version 4.2*x*, a S1000 error "Partial insert/update. The insert/update of a text or image column(s) did not succeed" is returned if the update affects more than one row.
String Function Parameters	*string_exp* parameters to the string functions must be of type SQL_CHAR or SQL_VARCHAR. SQL_LONG_VARCHAR types are not supported in the string functions. The *count* parameter must be less than or equal to 255, since the SQL_CHAR and SQL_VARCHAR datatypes are limited to a maximum length of 255 characters.
Time literals	Time literals, when stored in an SQL_TIMESTAMP column (SQL Server types of datetime or smalldatetime), have a date value of January 1, 1900.
timestamp	Only a Null value can be manually inserted into a timestamp column. However, because timestamp columns are automatically updated by SQL Server, a Null value is overwritten.
tinyint	The SQL Server tinyint datatype is unsigned. A *tinyint* column is bound to a variable of type SQL_C_UTINYINT by default.
User-defined datatypes	When connected to SQL Server version 4.2*x*, the SQL Server driver adds NULL to a column definition that does not explicitly declare a column's nullability. Therefore, the nullability stored

	in the definition of a user-defined datatype is ignored. When connected to SQL Server version 4.2*x*, columns with a user-defined datatype that has a base type of char / binary and for which no nullability is declared are created as type varchar / varbinary. **SQLColAttributes**, **SQLColumns**, and **SQLDescribeCol** return SQL_VARCHAR / SQL_VARBINARY as the datatype for these columns. Data retrieved from these columns is not padded.
LONG datatypes	SQL_LONGVARBINARY data must be passed to **SQLPutData** as raw binary data, not as binary data converted to character data. Also, data-at-execution parameters are restricted for both the SQL_LONGVARBINARY and SQL_LONGVARCHAR datatypes. For details, see the Data-at-Execution Parameter Limitations topic.

Nullability Resolution

In the ODBC grammar, columns for which nullability is not specified are assumed to be nullable. In the SQL Server grammar, columns for which nullability is not specified are assumed to be not nullable.

When connected to SQL Server 6.0, the SQL Server driver resolves this situation by using the SET ANSI_NULL_DEFAULT_ON ON SQL statement. This causes the server to use a default of NULL for columns where nullability is not specified except for bit columns or user-defined datatypes.

When connected to SQL Server version 4.2*x*, the SQL Server driver adds a Null specification to each column definition in a CREATE TABLE statement that does not specify whether the column is nullable (except for BIT columns, which are not nullable). It adds a NULL specification to each column definition in an ALTER TABLE statement (except for BIT columns, which are not nullable).

Using ODBC Cursors

SQL Server 6.0 provides support for engine-based cursors and version 2.5 of the SQL Server driver has been enhanced to make use of that powerful feature. This section provides a brief overview on how to use server cursors through the ODBC API.

ODBC Cursor Model

ODBC supports a cursor model allowing for several types of cursors, scrolling within a cursor, several concurrency options and single row as well as bulk fetches. Unlike ANSI SQL, ODBC does not have explicit APIs to Declare, Open and Close cursors. Cursors are automatically created for you when you issue a SQLExecDirect or SQLPrepare + SQLExecute based on the specified statement options.

Type A and Type B Statements

The SQL Server driver uses two types of statements Type A and Type B.

A Type A statement is defined as any statement using the default settings on each of the following three statement options.

Option	Default
SQL_CONCURRENCY	SQL_CONCUR_READ_ONLY
SQL_CURSOR_TYPE	SQL_CURSOR_FORWARD_ONLY
SQL_ROWSET_SIZE	1

Type A statements do not make use of server cursors. Instead the query is executed at the server and result sets are passed back to the application. The SQL Server driver allows you to perform **SQLFetch** and **SQLExtendedFetch** (you can change the rowset size after the cursor is open and perform block fetches) on this data, but the connection between the client and server remains busy until all the data has been fetched (youve reached the end of the cursor) or the cursor is closed. This option is fast if youre going to retrieve all rows sequentially and you dont need to perform any updates. Because server cursors are not used, you cannot do positioned operations (using WHERE CURRENT OF) on Type A statements.

A Type B statement is defined as any statement not using the default settings on the above three statement options. Type B statements cause the driver to use server cursors in most cases (please see the restrictions under Creating Cursors).

Advantages of Using Server Cursors

There are several advantages of using server based cursors implemented in SQL Server 6.0.

- Performance: If you are going to access a small fraction of the data in the cursor (typical of many browsing applications), using server based cursors will give you a big performance boost since only the required data (and not the entire result set) is sent over the network.
- Additional cursor types: Keyset and dynamic cursor types are only available if you use server based cursors. In addition, the cursor library only supports forward only with the read only concurrency and static with read only and optimistic concurrency. With server based cursors, you can use the full range of concurrency values with the different cursor types.
- Cleaner semantics: The cursor library simulates cursor updates by generating a SQL searched update statement. This can sometimes lead to unintended updates.
- Memory usage: When using server based cursors, the client does not need to cache large amounts of data or maintain information about the cursor position; the server provides that functionality.
- Multiple open cursors: When using server based cursors, the connection between the client and server does not remain busy between cursor operations. This allows you to have multiple cursors *hstmts* active at the same time. As per the ODBC specification, you are still restricted to one cursor per *hstmt*. To get around the one active *hstmt* limitation associated with Type A statements, change the statement options so you get server cursors associated with Type B statements.

See Also

Creating Cursors

Retrieving Data From Cursors

Updating Cursors

Closing Cursors

Creating Cursors

For the purposes of ODBC cursors, the cursor definition is essentially the SQL statement passed as an argument to **SQLExecuteDirect** or **SQLPrepare**/**SQLExecute**. The following rules apply when creating cursors through the SQL Server driver.

- Server cursors are created only for statements that begin with a SELECT, EXEC[ute] *procedure_name*, or {call *procedure_name*} clause. Otherwise, it is treated as a Type A statement. Server cursors are not created for a SQL batch. They are treated as Type A statements.
- If you ask for a Dynamic or Keyset cursor and there is not a unique index for every table referenced in the cursor definition, the server creates a static cursor and the driver returns a SQL_SUCCESS_WITH_INFO status.
- SQL Server allows you to create cursors on base tables as well as views. In addition, cursors can be opened on a query involving joins between multiple tables and/or aggregate operators. You cannot, however, use the keywords COMPUTE, COMPUTE BY, FOR BROWSE and INTO in the cursor definition.
- If the cursor definition involves a UNION, UNION ALL, outer join, GROUP BY with/without HAVING or DISTINCT, a Static cursor is always created irrespective of what was requested and the Driver returns an SQL_SUCCESS_WITH_INFO status.
- If you ask for a Dynamic cursor and the cursor definition contains an ORDER BY clause that does not match a unique index or a subquery, the server creates a Keyset cursor and the driver returns an SQL_SUCCESS_WITH_INFO status.
- If you ask for a concurrency of row versioning and there is not a TIMESTAMP column for every table in the cursor definition, the server uses a concurrency based on values and the driver returns an SQL_SUCCESS_WITH_INFO status.
- You will get a cursor on **SQLExecDirect** (Exec *procedure_name* or {Call *procedure_name*}) only if the procedure contains one SELECT statement and nothing else. Otherwise, SQL Server generates an error message. Because of this restriction, you cannot use server cursors with the ODBC catalog functions (which use stored procedures that contain multiple SELECT statements).
- You will automatically get the cursor behavior associated with Type A statements when executing a batch containing multiple SELECT statements (except if the first statement is a SELECT, EXECUTE, or ODBC canonical procedure invocation, in which case, SQL Server driver generates an error message).

Keyset cursors could also be populated asynchronously (please refer to the SQL Server server documentation on **sp_configure**). Asynchronous keyset population does not affect any of the cursor related ODBC APIs except for **SQLRowCount**.

The following topics discuss relevant ODBC functions for creating cursors.

SQLSetStmtOption

SQLSetScrollOptions

SQLSetConnectOption

SQLSetCursorName

SQLGetCursorName

SQLGetStmtOption

SQLGetInfo

SQLSetStmtOption

The SQLSetStmtOption function allows the caller to set various options at the statement level. The following cursor related options are supported.

Option	Value
SQL_CONCURRENCY	SQL_CONCUR_READ_ONLY SQL_CONCUR_LOCKING SQL_CONCUR_ROWVER SQL_CONCUR_VALUES
SQL_CURSOR_TYPE	SQL_CURSOR_FORWARD_ONLY SQL_CURSOR_STATIC SQL_CURSOR_KEYSET_DRIVEN SQL_CURSOR_DYNAMIC
SQL_ROWSET_SIZE	Integer value that specifies the number of rows in the rowset, i.e. the number of rows returned by each call to **SQLExtendedFetch**. The default value is 1.
SQL_BIND_TYPE	Integer value that sets the binding orientation to be used when **SQLExtendedFetch** is called on the associated *hstmt*. Both column and row bindings are supported.

SQL Server supports read-only static cursors and these are exposed through the SQL Server driver. Applications requiring updatable static cursors should choose the Cursor Library implementation by setting SQL_ODBC_CURSORS to SQL_CUR_USE_ODBC. Cursors provided by the Cursor Library are implemented at the client. All the data is cached and updates and deletes are issued by constructing a WHERE clause using the cached values.

Cursors implemented at the server (Forward Only, Keyset, Dynamic, Read Only Static) may be specified with a concurrency control option of read only, lock, row version or row values. Cursors implemented by the cursor library may be read only or row values.

Mixed cursors are not supported. The SQL_KEYSET_SIZE value is automatically converted to 0 if a non-zero value is specified.

SQLSetScrollOptions

The **SQLSetScrollOptions** function allows the caller to set options that control cursor behavior. In ODBC 2.0, these options have been superseded by the corresponding options in the **SQLSetStmtOption** function and are only supported for backward compatibility. The options fall into three categories as follows:

Option	**Value**
Concurrency Control	SQL_CONCUR_READ_ONLY SQL_CONCUR_LOCK SQL_CONCUR_ROWVER SQL_CONCUR_VALUES
Sensitivity	SQL_CURSOR_FORWARD_ONLY SQL_CURSOR_STATIC SQL_CURSOR_KEYSET_DRIVEN SQL_CURSOR_DYNAMIC
Rowset Size	Integer value

SQLSetConnectOption

The **SQLSetConnectOption** function allows the caller to set options that govern aspects of the connection.

The following cursor related options are supported.

Option	Value
SQL_ODBC_CURSORS	SQL_CUR_USE_IF_NEEDED SQL_CUR_USE_ODBC SQL_CUR_USE_DRIVER
SQL_TXN_ISOLATION	SQL_TXN_READ_UNCOMMITTED SQL_TXN_READ_COMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE SQL_TXN_VERSIONING option is not supported (In SQL Server, repeatable read is treated as serializable.)
Driver specific option	SQL_PRESERVE_CURSORS that allows you to choose between closing or preserving the cursor state across transaction commits/rollbacks. The default is to close cursors on a transaction commit/rollback.

SQLSetCursorName

The **SQLSetCursorName** function associates a cursor name with an active *hstmt*. A cursor name is automatically created if one is not specified.

SQLGetCursorName

The **SQLGetCursorName** function returns the cursor name associated with a specified *hstmt*. The cursor name can be user specified or system generated. This function can only be called if **SQLSetCursorName** has been previously called or after the cursor has been opened (by issuing an **ExecuteDirect**, **Execute**, or **Catalog** operation). The cursor name is valid for the lifetime of the *hstmt*, i.e. if the cursor is closed and another one is opened, the driver re-uses the previous name. Cursor names are required for issuing positioned updates and deletes.

SQLGetStmtOption

The **SQLGetStmtOption** function returns the current setting of options specified in **SQLSetStmtOption** plus a couple of additional flags. SQL_ROW_NUMBER specifies the number of the current row in the entire results set. This option is supported for keyset and static cursors; otherwise, 0 is returned.

SQLGetInfo

The **SQLGetInfo** function returns information about a specific driver and data source. For values provided by the SQL Server driver, see the SQLGetInfo Return Values topic.

Retrieving Data From Cursors

There are two ways to retrieve data through cursors.

- You can bind columns of the result set to storage locations using **SQLBindCol** and then use **SQLFetch** to move to the next row in the results set. For bulk fetches, you could bind columns to an array and use **SQLExtendedFetch**. The SQL Server driver supports row-wise as well as column-wise binding. Keep in mind that when using server cursors, each Fetch or Extended Fetch operation involves a round trip to the server and the connection remains free between these operations. The server keeps track of all required state information (for example, current cursor position). Also, cursor operations do not have to be sequential. The server allows you to traverse the cursor in the forward or backward directions relative to the current position. You can also directly position to an absolute row number (except for dynamic cursors) in the results set.
- You can retrieve data for unbound columns (columns for which storage has not been allocated). Use **SQLFetch** or **SQLExtendedFetch** to position the cursor on the next row and call **SQLGetData** to retrieve data for specific unbound columns. You can call **SQLGetData** multiple times on the same column to incrementally read in data. This is useful if you are reading large amounts of data from a text or image column. Keep in mind, however, that the connection is busy until you read the entire data associated with the column. If you issue an **SQLSetPos** with the Position option, the SQL Server driver flushes the remaining data and frees up the connection, thereby allowing you to use it for processing on another *hstmt*.

You can retrieve data from both bound and unbound columns in the same row.

When retrieving rows involving potentially large *text* and *image* columns, if you do not want to access the text data field in the row, dont bind that column. When you do an Extended Fetch, the retrieved row will only contain the *textptr* for the unbound column as opposed to the previous behavior of returning the complete row including the text data.

The following topics discuss relevant ODBC functions:

SQLFetch

SQLExtendedFetch

SQLRowCount

SQLFetch

The **SQLFetch** function fetches one row of data from the result set. It cannot be mixed with **SQLExtendedFetch** for the same cursor. Also, **SQLFetch** is forward only.

SQLExtendedFetch

The **SQLExtendedFetch** function returns one rowset (set in **SQLSetStmtOption**) of data to the application. It takes as input the statement handle, fetch type, and the number of the row to fetch. It returns the number of rows actually fetched and optionally, an array containing status values for each row.

The following fetch types are supported:

SQL_FETCH_NEXT
(required when cursor type is SQL_CURSOR_FORWARD_ONLY)

SQL_FETCH_FIRST

SQL_FETCH_LAST

SQL_FETCH_PRIOR

SQL_FETCH_ABSOLUTE
(not supported for dynamic cursors)

SQL_FETCH_RELATIVE

The SQL_FETCH_BOOKMARK and SQL_FETCH_RESUME types are not supported.

SQLRowCount

If this function is called after the cursor has been opened and the cursor is of type keyset or static, the **SQLRowCount** function returns the number of rows in the result set. For dynamic cursors **SQLRowCount** always returns 1. If the keyset is being populated asynchronously (please refer to the SQL Server documentation on **sp_configure**) **SQLRowCount** will return 1 until the keyset is fully populated.

Updating Cursors

Data is usually modified one row at a time (there is an option in **SQLSetPos** to affect the entire rowset) by first positioning to the desired row. To position the cursor on a particular row, you can:

- call **SQLFetch** one or more times.
- call **SQLExtendedFetch** one or more times.
- call **SQLSetPos** with the SQL_POSITION option.

There are two ways of modifying data in the results set.

- You can issue an UPDATE .... WHERE CURRENT OF *cursor_name* or DELETE WHERE CURRENT OF *cursor_name* to update or delete the row currently pointed to by the cursor. The positioned UPDATE or DELETE statement must be issued on a separate *hstmt* on the same connection and requires the cursor name (which you can explicitly set using the **SQLSetCursorName** function). This allows you to work with several cursors simultaneously.
- You can use the **SQLSetPos** function to add, delete and update rows of data. The **SQLSetPos** API provides options to specify the desired operation, the target row number and how to lock the row. **SQLSetPos** can only be used on rowsets fetched with **SQLExtendedFetch**.

The following topics discuss relevant ODBC functions:

SQLSetPos

SQLRowCount

SQLSetPos

The **SQLSetPos** function allows an application to set the cursor position in a rowset and perform an operation based on that position. The function takes as input the statement handle, position of the row in the rowset (on which the operation is to be performed), the operation type and an additional parameter describing how to lock the row after performing the operation.

The *irow* argument specifies the number of the row (in the rowset) on which to perform the operation. If *irow* = 0, the operation applies to the entire rowset.

The following operations are supported:

SQL_POSITION

SQL_REFRESH

SQL_UPDATE

SQL_DELETE

SQL_ADD

The following lock types are supported:

SQL_LOCK_NO_CHANGE

Note that if a row is marked as deleted (SQL_ROW_DELETED), and later a row is inserted with the same key value(s) as the old row, a SQL_REFRESH operation will show the new row (SQL_ROW_SUCCESS) in the old position.

SQLRowCount

The **SQLRowCount** function returns the number of rows affected by an UPDATE, INSERT or DELETE statement or by the SQL_UPDATE, SQL_ADD, or SQL_DELETE operations in **SQLSetPos**. **SQLRowCount** can be called after a database update has been performed.

Closing Cursors

Cursors are automatically closed if you commit or rollback the transaction. The SQL Server driver provides a driver-specific connection option, SQL_PRESERVE_CURSORS,   to override this behavior for server cursors. If this option is set to SQL_PC_ON, cursors remain open and the cursor state is preserved across transaction commits or rollbacks.

The following topics discuss relevant ODBC functions:

SQLFreeStmt

SQLTransact

SQLFreeStmt

The **SQLFreeStmt** function stops processing associated with a specific statement, closes any open cursors, discards pending results and optionally frees all resources associated with the statement handle.

The following cursor related options are supported:

SQL_CLOSE

SQL_DROP

SQLTransact

The **SQLTransact** function requests a commit or rollback operation for all active operations on all statements associated with the connection. All open server cursors remain open after this operation if SQL_PRESERVE_CURSORS has been set to SQL_PC_ON. By default, cursors are closed on Commit/Rollback.

ODBC API Function Implementation

The SQL Server driver supports translation DLLs as well as the functions listed here. For an explanation of how each supported function is implemented, click on the function.

Function	Description
SQLBrowseConnect	**SQLBrowseConnect** uses three levels of keywords: 1. DSN, DRIVER 2. SERVER, UID, PWD, APP, and WSID 3. DATABASE and LANGUAGE
SQLConnect	**SQLConnect** supports DATABASE and LANGUAGE defaults.
SQLDriverConnect	**SQLDriverConnect** uses the DSN, DRIVER, SERVER, UID, PWD, APP, WSID, DATABASE, and LANGUAGE keywords.
SQLColAttributes, SQLDescribeCol, and SQLNumResultCols	If any of these functions are called after a SELECT statement has been prepared and before it has been executed, the SQL Server driver uses SET FMTONLY to cause SQL Server to generate the necessary information about the result set.
SQLConfigDataSource	**SQLConfigDataSource** adds, modifies, or deletes a data source dynamically by using keywords to set connect options.
SQLPrepare	SQL Server doesn't directly support the Prepare/Execute model of ODBC. To prepare an SQL statement, the SQL Server driver creates a temporary procedure which compiles it for later execution. Note that generation of stored procedures for **SQLPrepare** can be disabled in either the ODBC SQL Server Driver Setup dialog box or the **SQLSetConnectOption** function. If disabled, the SQL statement is stored sent to the server each it's executed.
SQLParamOptions	The **SQLParamOptions** function allows an application to specify an array of multiple values for the parameters assigned by **SQLBindParameter**. By calling **SQLParamOptions** with a *crow* greater than 1, the driver generates a SQL statement batch or a stored procedure batch to execute multiple SQL statements.
SQLDescribeParam	The **SQLDescribeParam** function returns the description of a parameter marker associated with a prepared SQL statement.
SQLRowCount	If this function is called after a cursor open and the cursor is of type keyset or static, the function returns the number of rows in the returns the number of rows in the result set. For dynamic cursors **SQLRowCount** always returns 1. If the keyset is being populated asynchronously **SQLRowCount** will return 1 until the keyset is fully populated.
SQLTablePrivileges	The driver does not support search patterns for the table owner and table name parameters.
SQLTables	The driver does not support search patterns for the table qualifier parameter.

SQLBrowseConnect

SQLBrowseConnect uses three levels of connection information. For each keyword, the following table indicates whether a list of valid values is returned, and whether the keyword is optional. For a description of the keyword, click on the keyword.

Level	Keyword	List of Valid Values Returned?	Optional?	Description
1	**DSN**	N/A	No	The name of the data source as returned by **SQLDataSources**. The DSN keyword is not used if DRIVER is used.
	DRIVER	N/A	No	The name of the driver as returned by **SQLDrivers**. The DRIVER keyword is not used if DSN is used. The SQL Server driver name is {SQL Server}. (For a 16-bit program the 32-bit driver name is {SQL Server (32 bit)}.)
2	**SERVER**	Yes	No	The name of the server on the network on which the data source resides. When running on Microsoft Windows NT, "(local)" can be entered as the server, in which case a local copy of SQL Server can be used, even when this is a non-networked version. Note that when the 16-bit SQL Server driver is using "(local)" without a network, the "MS Loopback Adapter" must be installed.
	UID	No	Yes	The user login ID.
	PWD	No	Yes (depends on the user)	The user-specified password.
	APP	No	Yes	The name of the application (AppName) calling **SQLBrowseConnect**.
	WSID	No	Yes	The workstation ID. Typically, this is the network name of the

				computer on which the application resides.
3	**DATABASE**	Yes	Yes	The name of the SQL Server database.
	LANGUAGE	Yes	Yes	The national language to be used by SQL Server.

SQLBrowseConnect ignores the values of the **Database** and **Language** keywords stored in the ODBC data source definitions. If the database or language specified in the connection string passed to **SQLBrowseConnect** is invalid, **SQLBrowseConnect** returns SQL_NEED_DATA and the level 3 connection attributes.

SQLBrowseConnect doesn't check user access to all the databases listed with the **DATABASE** keyword. If the user doesn't have access to the chosen database, **SQLBrowseConnect** returns SQL_NEED_DATA and the level 3 connection attributes.

See Also

SQLConnect

SQLDriverConnect

SQLConnect

The **SQLConnect** function uses the following keywords:

Parameter	Implementation
DSN	The name of the data source as returned by **SQLDataSources**.
UID	The user login ID.
AuthStr	The user-specified authentication string (typically the password).

SQLConnect retrieves the value of the LANGUAGE keyword from the ODBC data source definition. If the SQL Server driver is unable to use the specified language, it uses the default language for the specified user ID, and **SQLConnect** returns SQL_SUCCESS_WITH_INFO. If SQL Server is unable to use the default language for the specified user ID, **SQLConnect** returns SQL_ERROR.

SQLConnect retrieves the value of the DATABASE keyword from the ODBC data source definition. If the SQL Server driver is unable to use the specified database, it uses the default database for the specified user ID, and **SQLConnect** returns SQL_SUCCESS_WITH_INFO. If SQL Server is unable to use the default database for the specified user ID, **SQLConnect** returns SQL_ERROR.

See Also

SQLBrowseConnect

SQLDriverConnect

SQLDriverConnect

The **SQLDriverConnect** connection string uses the following keywords:

Keyword	Description
DSN	The name of the data source as returned by **SQLDataSources**. The DSN keyword is not used if DRIVER is used.
DRIVER	The name of the driver as returned by **SQLDrivers**. The DRIVER keyword is not used if DSN is used. The SQL Server driver name is {SQL Server}.
SERVER	The name of the server on the network on which the data source resides. On a Microsoft Windows NT computer, "(local)" can be entered as the server, in which case a local copy of SQL Server can be used, even when this is a non-networked version. Note that when the 16-bit SQL Server driver is using "(local)" without a network, the "MS Loopback Adapter" must be installed.
UID	The user login ID.
PWD	The user-specified password.
APP	The name of the application calling **SQLDriverConnect** (optional).
WSID	The workstation ID. Typically, this is the network name of the computer on which the application resides (optional).
DATABASE	The name of the SQL Server database (optional).
LANGUAGE	The national language to be used by SQL Server (optional).

SQLDriverConnect uses keyword values from the dialog box (if one is displayed). If a keyword value is not set in the dialog box, **SQLDriverConnect** uses the value from the connection string. If the value is not set in the connection string, it uses the value from the ODBC.INI file.

If the *fDriverCompletion* argument is SQL_DRIVER_NOPROMPT or SQL_DRIVER_COMPLETE_REQUIRED, the language or database comes from the connection string, and the language or database is invalid, **SQLDriverConnect** returns SQL_ERROR.

If the *fDriverCompletion* argument is SQL_DRIVER_NOPROMPT or SQL_DRIVER_COMPLETE_REQUIRED, the language or database comes from the ODBC data source definitions, and the language or database is invalid, **SQLDriverConnect** uses the default language or database for the specified user ID and returns SQL_SUCCESS_WITH_INFO.

If the *fDriverCompletion* argument is SQL_DRIVER_COMPLETE or SQL_DRIVER_PROMPT and the language or database is invalid, **SQLDriverConnect** redisplays the dialog box.

For example, to connect to the Human Resources data source on the server HRSERVER using the login ID Smith and the password Sesame, you would use the following connection string:

```
DSN=Human Resources;UID=Smith;PWD=Sesame
```

To specify the Payroll database on the same server, you would use the following connection string:

```
DSN=Human Resources;UID=Smith;PWD=Sesame;DATABASE=Payroll
```

To specify the SQL Server driver (the driver name is {SQL Server}) and server name directly without using a DSN, you would use the following connection string:

```
DRIVER={SQL Server};SERVER=hrserver;UID=Smith;PWD=Sesame;
    DATABASE=Payroll
```

See Also

SQLBrowseConnect

SQLConnect

SQLColAttributes, SQLDescribeCol, and SQLNumResultCols

Microsoft SQL Server returns information about a result set before it returns the data in the result set. The SQL Server driver returns this information to an application through the **SQLColAttributes**, **SQLDescribeCol**, and **SQLNumResultCols** functions.

When connected to SQL Server 6.0, the SQL Server driver uses the SET FMTONLY statement to retrieve the appropriate information about a result set.

When connected to SQL Server version 4.2*x*, if an application calls any of these functions after a SELECT statement has been prepared and before it has been executed, the SQL Server driver submits the SELECT statement with the clause WHERE 1=2. This forces SQL Server to generate a result set without any rows, but with the information about the result set.

Note When connected to SQL Server 4.2x, **SQLColAttributes**, **SQLDescribeCol**, and **SQLNumResultCols** cannot return information about a result set generated by a procedure if that procedure has been prepared but not executed. If the SELECT statement is the first statement in a batched statement and SQL Server native grammar is used (no semicolons between statements), the results of these functions are unpredictable. Note also that the word "SELECT" must be the first token in the buffer. If anything precedes the word "SELECT" in the statement to be prepared, "WHERE 1=2" will not be added to the SELECT statement and the information about the result set will not be returned.

SQLConfigDataSource

The **SQLConfigDataSource** function is used to add, modify, or delete a data source dynamically and uses the following keywords. Note that only the SERVER keyword is required for this function; all other keywords are optional.

Keyword	Description
ADDRESS	The network address of the SQL Server database management system from which the driver retrieves data.
DATABASE	The name of the SQL Server database.
DESCRIPTION	A description of the data in the data source.
LANGUAGE	The national language to be used by SQL Server.
NETWORK	The network library connecting the platforms on which SQL Server and the SQL Server driver reside.
OEMTOANSI	Enables conversion of the OEM character set to the ANSI character set if the SQL Server client machine and SQL Server are using the same non-ANSI character set. Valid values are YES for on (conversion is enabled) and NO for off. The default value that's set using the Client Configuration Tool.
SERVER	The name of the network computer on which the data source resides.
TRANSLATIONDLL	The name of the DLL that translates data passing between an application and a data source.
TRANSLATIONNAME	The name of the translator that translates data passing between an application and a data source.
TRANSLATIONOPTION	Enables translation of data passing between an application and a data source.
USEPROCFORPREPARE	Disables generation of stored procedures for SQLPrepare. Valid values are NO for off (generation is disabled) and YES for on. The default value (set in the setup dialog box) is YES.

Note that keyword pairs in **SQLConfigDataSource** strings are null-byte (\0) terminated, and the string itself is also null-byte (\0) terminated. For example:

```
DSN=Human Resources\0SERVER=hrserver\0ADDRESS=hrnetaddr\0
     NETWORK=DBMSSOCN\0DATABASE=payroll\0\0
```

SQLPrepare

SQL Server doesn't directly support the Prepare/Execute model of ODBC.

First, a temporary stored procedure is created from the statement, since stored procedures are an efficient way to execute a statement more than once. The procedure is named "#odbc#*useridentifier*", where *user* is up to 6 characters of the username and *identifier* is up to 8 digits and identifies the statement. The procedure is created at prepare time if all parameters have been set, or at execute time if all parameters were not set at prepare time or if any parameter has been reset since the procedure was created. Because of this, **SQLExecute** can return any errors that **SQLPrepare** can return.

If the CREATE PROCEDURE returns an error, **SQLPrepare** submits the statement to SQL Server with the SET NOEXEC or SET PARSEONLY option (depending on the statement type). SQL Server checks the syntax of the statement and returns any errors.

If a user cannot create a stored procedure for any reason (such as lack of permission), the SQL Server driver doesn't use a stored procedure but submits the SQL statement each time **SQLExecute** is called.

You can disable a generation of stored procedures for **SQLPrepare**. If disabled, the statement will be stored and re-executed at execution time. Stored procedure generation can be disabled You can disable in either of two ways:

- Clearing the Generate Stored Procedures for Prepared Statements option checkbox in the ODBC SQL Server Driver Setup dialog box.
- Setting the SQL_USE_PROCEDURE_FOR_PREPARE option in the **SQLSetConnectOption** function to SQL_UP_OFF.

When stored procedure generation is disabled and the statement is stored and executed at run time, all syntax error checking is delayed until run time.

Note that if SET NOCOUNT ON has been executed, multiple statements embedded in a stored procedure don't create multiple results as they should. Row counts generated by SQL statements inside of a stored procedure are ignored by the driver.

SQLParamOptions

The **SQLParamOptions** function allows an application to specify an array of multiple values for the parameters assigned by **SQLBindParameter**. By calling **SQLParamOptions** with a *crow* greater than 1, the driver generates a SQL statement batch or a stored procedure batch to execute multiple SQL statements.

SQLParamOptions is usually used with DML. Any DDL statements used with **SQLParamOptions** will have the same result as executing the statement serially *crow* times. For statements that do not contain parameter markers, the statement will still be executed *crow* times.

A stored procedure batch is used for **SQLPrepare** / **SQLExecute** and canonical procedure invocations that don't use any data-at-execution parameters. Stored procedure output parameters are returned.

A SQL batch (also called a language event) is used for SQL commands and any canonical procedure invocations that use data-at-execution parameters. Any stored procedure output parameters are not returned. If a SELECT statement is used, multiple results sets will be returned, and the application must use **SQLMoreResults** to process them. After **SQLParamData**, **SQLExecDirect**, or **SQLExecute** returns success, the value returned in *pirow* is initially set to 1, and **SQLMoreResults** increments this value as you process each results set.

SQLRowCount will return the total of all rows affected by all statements in the SQL or stored procedure batch that contains no SELECT statements. The value returned by **SQLRowCount** is undefined if one or more SELECT statements are included. Also, when the original SQL contains multiple statements the value returned in *pirow* can be incorrect.

While processing a batch, SQL Server may return an error. In some cases, SQL Server cancels the current command only, and the remainder of the batch is processed. For these errors, one or more executions will fail, and **SQLExecDirect**, **SQLExecute** or **SQLParamData** will return SQL_ERROR. The value returned in *pirow* will be set to *crow* because all parameter rows were processed. **SQLError** will return the errors that occurred, but there is no way to determine the parameter row(s) which caused the error. Because all parameter rows were processed, re-binding and continuing execution is not necessary.

In other cases, an error causes SQL Server to cancel the entire batch. For these errors, the value returned in *pirow* will contain the affected parameter row. You must re-bind and continue execution to process the remaining parameter rows.

When data truncation occurs for a parameter, processing continues for the remaining parameters. SQL_SUCCESS_WITH_INFO is returned, along with a warning message. However, when an error (such as a conversion error) occurs on the client for one of the parameters the entire batch execution is stopped, and no execution is done for any row.

While **SQLParamData** returns SQL_NEED_DATA, it updates the value of *pirow* as it processes data-at-exec parameters, and the final **SQLParamData** updates the value of *pirow* based on any errors encountered during execution. Thus the value in *pirow* may be larger during the data-at-exec parameter processing than it is after the execution has completed if an error is encountered with a row during execution.

When connected to SQL Server 4.2*x*, **SQLParamOptions** has the following limitations:

- SQL Server 4.2*x* is limited to approximately 128,000 bytes for a single language event. If using **SQLParamOptions** causes this limit to be exceeded, the driver will return the SQL Server error message.
- SQL Server 4.2*x* does not support stored procedure batches. Thus, any stored procedure output parameters are not returned.

Note that when you pass a non-NULL value for *pirow*, the driver will continue to place the current row number in that memory address, even if **SQLParamOptions** is effectively "turned off" by passing a

crow of 1. Thus, if the *pirow* address supplied by the application becomes an invalid address (for example, a local variable leaving scope) it is possible that the driver might fail.

SQLDescribeParam

The **SQLDescribeParam** function returns the description of a parameter marker associated with a prepared SQL statement.

Parameter markers must appear in the correct ODBC locations, as specified in the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*. Also, the driver does not support the following items:

- Calling **SQLDescribeParam** after execution.
- Calling **SQLDescribeParam** for stored procedure calls. It is recommended that **SQLProcedureColumns** be used for stored procedures.
- The Transact-SQL FROM clause extension to UPDATE and DELETE statements.
- Parameter markers in a sub query. The driver will return an error.

SQL Support

The SQL Server driver fully supports the core SQL grammar, with the exception of the USER keyword. The SQL Server driver also supports almost all SQL statements in the extended ODBC grammar. In accordance with the design of ODBC, the SQL Server driver will pass native SQL grammar to SQL Server.

For details on the implementation and limitations of ODBC SQL grammar by the SQL Server driver, see the following Help topics.

SQL Grammar Limitations

Non-supported ODBC SQL Grammar

Nullability Resolution

SQL Grammar Limitations

The SQL Server driver and SQL Server impose the following programming limitations on the ODBC SQL grammar:

SQL Grammar	Limitation
Data-at-Execution Parameter Limitations	Data-at-execution parameters that are used to send more than 65,535 bytes of data for an SQL_LONGVARCHAR or SQL_LONGVARBINARY column are subject to a number of restrictions. For details, see the Data-at-Execution Parameter Limitations topic.
OUTER JOIN	The search condition for an outer join must be an equals condition (=). The ODBC outer join extension must appear last in the table reference list. SQL Server does not support both nested outer joins and inner joins nested within an outer join. A table cannot be the inner table in an outer join and still participate in an inner join. A table cannot be the inner table in an outer join and the outer table in another outer join. A table can, however, participate in an inner join and be the outer table in an outer join.
Procedures	A procedure must be invoked as the first statement in a prepared batch of statements or it must be invoked with the EXECUTE keyword or using the ODBC canonical procedure innovation. For details, see the Procedure Invocation Limitations topic.

Data-at-Execution Parameter Limitations

If you use a data-at-execution parameter to send more than 65,535 bytes of data for an SQL_LONGVARCHAR (text) or SQL_LONGVARBINARY (image) column, it is subject to the following restrictions:

- It can be used only as an *insert_value* in an INSERT statement.
- It can be used only as an *expression* in the SET clause of an UPDATE statement.
- It cannot be used in a statement with data-at-execution parameters that have a datatype other than SQL_LONGVARCHAR or SQL_LONGVARBINARY.
- It can be used in a statement with non-data-at-execution parameters of any datatype.
- If its value is NULL, the *cbColDef* argument in **SQLBindParameter** and the *cbValueMax* argument in **SQLPutData** must be SQL_NULL_DATA.
- Canceling the operation (calls to **SQLPutData** for a column after **SQLParamData** returns SQL_NEED_DATA) before completion will cause a partial update. The text or image column you were processing when you canceled will be set to an intermediate "place holder" value, and any unprocessed text or image columns remain unchanged.

Procedure Invocation Limitations

If a statement that executes a procedure is the first statement in a batch, the EXECUTE keyword is not needed. If such a statement is not the first statement, the EXECUTE keyword must be present. This is because the SQL Server driver surrounds a statement it is preparing with other SQL statements. Thus, the statement being prepared is no longer the first in the batch.

For maximum interoperability, procedures should be invoked using the ODBC extension to the SQL designed for this purpose. With the SQL Server driver, there is no advantage to preparing a statement that invokes a procedure (instead of executing it directly). When the Generate Stored Procedures for Prepared Statements option is selected, the SQL Server driver prepares a statement by placing it in a procedure and compiling that procedure.

Preparing a statement by placing it in a procedure and compiling can be disabled, however. To ensure that stored procedures are never used to implement **SQLPrepare**, clear the Generate Stored Procedures option checkbox in the SQL Server Setup Dialog Box, or set the SQL_USE_PROCEDURE_FOR_PREPARE option in **SQLSetConnectOption** to SQL_UP_OFF. This will ensure that a prepared statement will be stored and executed at run time. Stored procedures will then not be used to implement **SQLPrepare**. In addition, all syntax error checking will be delayed until execution time.

When a SET NOCOUNT ON statement is executed, multiple statements embedded in a stored procedure do not create multiple results as they should. The driver ignores row counts generated by SQL statements inside of a stored procedure.

Literals as Parameters to Stored Procedure Calls

Note that a literal of NULL cannot be passed to a stored procedure parameter defined as *datetime* or *smalldatetime*.

Cursors Using Stored Procedures

Server cursors can only be opened on a stored procedure that contains a single SELECT statement. Using a stored procedure that contains no SELECT statements, or more than one SELECT statement, will generate an error. Because of this restriction, server cursors cannot be used with the ODBC catalog functions.

Non-supported ODBC SQL Grammar

The SQL Server driver fully supports all core and extended ODBC SQL grammar with the following exceptions.

Non-supported Grammar	**Description**
DAYOFWEEK	The scalar function DAYOFWEEK is not supported when connected to 4.2x. The function is supported when connected to SQL Server 6.0.
DELETE	The WHERE CURRENT OF *cursor_name* clause (positioned delete statement) is not supported when connected to SQL Server version 4.2*x*.
DROP INDEX	*table_name*, *index_name* must be used instead of just *index_name*.
IEF	None of the clauses in the Integrity Enhancement Facility (IEF) are supported when connected to SQL Server version 4.2*x*.
MAX, MIN	The DISTINCT keyword is not supported for these set functions.
SELECT	The FOR UPDATE OF clause (SELECT-FOR-UPDATE statement) is not supported when connected to SQL Server version 4.2*x*.
UPDATE	The WHERE CURRENT OF *cursor_name* clause (positioned UPDATE statement) is not supported when connected to SQL Server version 4.2*x*.
USER	The USER keyword is not supported, except in the DEFAULT clause of the IEF.

Active *hstmt*

The SQL Server driver can have only one active *hstmt* unless it is used with server cursors; it returns this information through **SQLGetInfo** with the SQL_ACTIVE_STATEMENTS option. An *hstmt* is defined as active if it has results pending. In this context, *results* are any information returned by SQL Server, such as a result set or a count of the rows affected by an UPDATE statement.

Note An *hstmt*'s activity is not related to its state. For example, if a SELECT statement is executed and it doesn't return any rows, the statement is not active, since no results are pending. However, before the statement can be executed again, the cursor associated with it must be closed with **SQLFreeStmt**.

Server cursors allow multiple active statements on a single connection when connected to SQL Server 6.0.

When connected to SQL Server version 4.2*x*, the SQL Server driver supports only one active statement per connection. The cursor library (shipped with the SQL Server driver) allows applications to use multiple active statements on a connection.

Cursor Library

The cursor library (shipped with the SQL Server driver) allows applications to use multiple active statements on a connection, and scrollable, updateable cursors. The cursor library (ODBCCURS.DLL (16-bit) or ODBCCR32.DLL (32-bit)) must be loaded in order to support this functionality. Use **SQLSetConnectOption** to specify how the cursor library should be used and **SQLSetStmtOption** to specify the cursor type, concurrency, and rowset size.

Arithmetic Errors

If an arithmetic error such as divide-by-zero or a numeric overflow occurs during execution of a SQL statement, the statement is aborted. Partial results may have already been sent to the client and maybe delivered to the application before the application is notified of the error.

The driver sets ARITHABORT to ON.

For more information, see SET ARITHABORT and SET ARITHIGNORE in the *Microsoft SQL Server Transact-SQL Reference*.

Manual-commit Mode Transactions

When the SQL Server driver is in manual-commit mode, it initiates a transaction with a BEGIN TRANSACTION statement in the following situations:

- An SQL statement is pending.
- There is no current transaction.
- It is not a restricted Data Definition Language (DDL) statement.

  For more information about restricted DDL statements, see the section on Transactions in the *Microsoft SQL Server Transact-SQL Reference* documentation for the list of statements that cannot be in a transaction.

To commit or roll back a transaction in manual-commit mode, the application must call **SQLTransact**. The SQL Server driver sends a COMMIT TRANSACTION statement to commit a transaction; it sends a ROLLBACK TRANSACTION statement to roll back a transaction.

A restricted DDL statement can be executed only in manual-commit mode under one of the following circumstances:

- After manual commit mode has been set and before a non-restricted DDL or a Data Manipulation Language (DML) statement has been executed, or
- After a transaction has been committed or rolled back and before a non-restricted DDL or a DML statement has been executed.

  For more information about manual-commit mode, see **SQLSetConnectOption** in the *Microsoft ODBC SDK Programmer's Reference*.

The following ODBC catalog functions cannot be called inside a transaction because they used catalog stored procedures that create temporary tables:

SQLColumnPrivileges
SQLForeignKeys
SQLPrimaryKeys (limited on SQL Server 4.2*x* only)
SQLStatistics
SQLTablePrivileges

Remote Procedure Calls

The SQL Server driver uses the remote procedure call (RPC) facility in SQL Server to run procedures rather than pass procedures to SQL Server in an SQL statement. A procedure can be a prepared statement (which is a statement that is stored as a procedure), a procedure called with the ODBC procedure syntax, or a stored procedure that the SQL Server driver uses to implement a catalog function. RPCs have the following advantages over procedures passed in an SQL statement:

- RPCs are faster than procedures passed in an SQL statement.
- RPCs can have output parameters (however, see "Limitations", below); procedures passed in an SQL statement cannot. (A procedure can return a value in either case.)

To run a statement as an RPC, an application

1. Constructs an SQL statement.
2. Calls SQLBindParameter for each parameter in the statement.
3. Prepares the statement with SQLPrepare. (Note that the SQL_USE_PROCEDURE_FOR_PREPARE connection option must be set to SQL_UP_ON.)
4. Executes the statement with SQLExecute.

To run a procedure as an RPC, an application

1. Constructs an SQL statement that uses the ODBC procedure syntax. The statement uses parameter markers for each input, input/output, and output parameter, and for the procedure return value (if any).
2. Calls **SQLBindParameter** for each input, input/output, and output parameter, and for the procedure return value (if any).
3. Executes the statement with **SQLExecDirect**.

Note If an application submits a procedure using the SQL Server syntax (as opposed to the ODBC procedure syntax), the SQL Server driver passes the procedure call to SQL Server as an SQL statement rather than as an RPC.

Limitations

RPCs are executed as language events (EXEC *Procname Params*) if any parameter is bound to using SQL_DATA_AT_EXEC.

When an RPC is executed as a language event, no output parameter values are returned.

Output parameters of a stored procedure procedure are also input parameters, although the stored procedure many not use the input value. However, a valid input value must be provided to prevent conversion errors.

For input/output parameters and output parameters, the length/precision (*cbColDef*) and scale (*ibScale*) set using **SQLBindParameter** limit the maximums that the output value can contain.

References

For more information on the ODBC SQL Server Driver, see the following resources:

Reference	Source
Microsoft ODBC 2.0 Programmer's Reference and SDK Guide	Microsoft Press
ODBC API Reference	SQL Server Books Online
SQL Server documentation	SQL Server Books Online

Glossary

API

Application programming interface. A set of routines available in an application for use by software programmers when designing an application.

character set

A character set is a set of 256 letters, numbers, and symbols specific to a country or language. Each character set is defined by a table called a code page. An OEM (Original Equipment Manufacturer) character set is any character set except the ANSI character set. The ANSI character set (code page 1252) is the character set used by Microsoft Windows.

conformance level

Some applications can use only drivers that support certain levels of functionality, or conformance levels. For example, an application might require that drivers be able to prompt the user for the password for a data source. This ability is part of the conformance level for the application programming interface (API).

Every ODBC driver conforms to one of three API levels (Core, Level 1, Level 2) and one of three SQL grammar levels (Minimum, Core, Extended). ODBC drivers can support some of the functionality in levels above their stated level.

For detailed information about conformance levels, see the *Microsoft ODBC SDK Programmer's Reference*.

data source

A data source contains data and the information needed to access that data. Examples of data sources are:

- A SQL Server database, the server on which it resides, and the network used to access that server.
- A directory containing a set of dBASE files.

DBMS

Database management system. The software used to organize, analyze, search for, update, and retrieve data.

DDL

Data definition language. Any SQL statement that can be used to define data objects and their attributes. Examples include CREATE TABLE, DROP VIEW, and GRANT statements.

DLL

Dynamic-link library. A set of routines that one or more applications can use to perform common tasks. The ODBC drivers are DLLs.

DML

Data manipulation language. Any SQL statement that can be used to manipulate data. Examples include UPDATE, INSERT, and DELETE statements.

ODBC

Open Database Connectivity. A Driver Manager and a set of ODBC drivers that enable applications to access data using SQL as a standard language.

ODBC Driver Manager

A dynamic-link library (DLL) that manages access to ODBC drivers.

ODBC driver

A dynamic-link library (DLL) that an ODBC-enabled application, such as Microsoft Excel, can use to gain access to a data source. Each database management system (DBMS), such as Microsoft SQL Server, requires a driver.

SQL

Structured Query Language. A language used to retrieve, update, and manage data.

SQL statement

A command written in Structured Query Language (SQL); also known as a query. An SQL statement specifies an operation to perform, such as SELECT, DELETE, or CREATE TABLE; the tables and columns on which to perform that operation; and any constraints to that operation.

translation option

An option that specifies how a translator translates data. For example, a translation option might specify the character sets between which a translator translates character data. It might also provide a key for encryption and decryption.

translator

A dynamic-link library (DLL) that translates all data passing between an application, such as Microsoft Access, and a data source. The most common use of a translator is to translate character data between different character sets. A translator can also perform tasks such as encryption and decryption or compression and decompression.