SYBASE®
An SAP® Company

Users Guide

# Adaptive Server® Enterprise ODBC Driver by Sybase 15.7 SP100

Microsoft Windows and UNIX

DOCUMENT ID: DC20116-01-1570100-01
LAST REVISED: May 2013

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# Contents

# ODBC Programming

Open Database Connectivity (ODBC) is a programming interface for developing applications to access data from database management systems.

The primary documentation for ODBC application development is the *Microsoft ODBC SDK* documentation at *http://msdn.microsoft.com*. The introductory material and feature descriptions specific to Adaptive Server® Enterprise ODBC Driver are provided, but is not a complete guide to ODBC application programming.

## ODBC Requirements and Supported Platforms

The ODBC interface is a call-based application programming interface defined by Microsoft Corporation as a standard interface to database management systems on Microsoft Windows.

In addition, ODBC is now widely used on many non-Windows platforms, such as Linux.

### *Software Requirements*

To write ODBC applications for Adaptive Server Enterprise, you need:

- Adaptive Server Enterprise
- A C compiler capable of creating programs for your environment
- ODBC Software Development Kit (SDK):
  - On Windows platform the operating system together with the Visual Studio compiler provide all required components. Alternatively, install the Microsoft Data Access Components (MDAC).
  - For non-Windows platforms, there are commercial and open source projects such as unixODBC and iODBC that provide the ODBC SDK, including the ODBC Driver Manager and header files. Linux operating system includes such open source distributions.
  - On HP HP-UX, IBM AIX, and Sun Solaris, you can use the iAnywhere ODBC Driver Manager, which is included in your Adaptive Server ODBC Driver installation. You can also use other commercial or open source distributions of the ODBC SDK, however, to do so, you must install them separately.

    **Note:** The iAnywhere ODBC Driver Manager does not map calls to ODBC versions 1.0 and 2.0 to calls to ODBC version 3.x. Applications using the iAnywhere ODBC Driver Manager must restrict their use of the ODBC feature set to versions 3.0 and later.

### *Supported Platforms*

See the *Open Server and SDK New Features for Microsoft Windows, Linux, and UNIX* for a list of platforms on which Adaptive Server ODBC Driver is available.

**Note:** Significant portions of this book deal with writing C programs to access data using ODBC functions with Adaptive Server ODBC Driver. There are utilities, programs, and 4GL RAD tools that can use ODBC connections. For example, you can write a PowerBuilder® application or a PHP Web page that connects to an ODBC datasource. For such uses, you only need to know how to set up a datasource using Adaptive Server ODBC Driver. Once the datasource has been set up, these tools completely abstract the underlying ODBC function calls.

## ODBC Conformance

The Adaptive Server ODBC Driver conforms to ODBC 3.52 specification.

ODBC features are arranged according to level of conformance. Features are either Core, Level 1, or Level 2, with Level 2 being the most complete level of ODBC support. These features are listed in the *Microsoft ODBC Programmer's Reference*.

The Adaptive Server ODBC Driver meets Level 2 conformance with these exceptions:

- Level 1 conformance – The Adaptive Server ODBC Driver supports all Level 1 features except SQLSetPos with SQL_REFRESH.
- Level 2 conformance – The Adaptive Server ODBC Driver supports all Level 2 features except for using bookmarks in SQLBulkOperations (SQL_FETCH_BY_BOOKMARK, SQL_UPDATE_BY_BOOKMARK, SQL_DELEET_BY_BOOKMARK).

Applications developed using older versions of ODBC continue to work with the Adaptive Server ODBC Driver and the newer ODBC Driver Manager. The new ODBC features are not available for older applications.

## ODBC Driver Manager

The ODBC Driver Manager manages the communications between the user applications and the ODBC drivers.

Typically, user applications are linked against the ODBC Driver Manager. The Driver Manager manages the job of loading and unloading the appropriate ODBC Driver for the application. Applications make ODBC calls to the ODBC Driver Manager, which performs basic error checking and then processes these calls or passes them on to the underlying ODBC Driver.

The ODBC Driver Manager is not a required component, but it exists to solve many issues surrounding ODBC application development and deployment. Some advantages of using an ODBC Driver Manager are:

- Portable data access. Applications do not need to be rebuilt to use a different database management systems (DBMS).

- Runtime binding to a datasource.
- Ability to easily change a datasource.

To use the Adaptive Server ODBC Driver without using the ODBC Driver Manager, link your application directly to the Adaptive Server ODBC Driver library. The resulting executable connects only to Adaptive Server datasources.

The Adaptive Server ODBC Driver has been tested with these ODBC Driver Managers:

- On Microsoft Windows – the Microsoft ODBC Driver Manager that is included with Microsoft Windows
- On Linux – the unixODBC Driver Manager that is included with Red Hat and SuSE
- On HP HP-UX, IBM AIX, and Solaris – the unixODBC Driver Manager version 2.2.14 and the Sybase iAnywhere ODBC Driver Manager included with the Adaptive Server ODBC Driver installation

**Note:** Historically, the unixODBC Driver Manager on 64-bit Linux platforms has expected a 4-byte `SQLLEN` from ODBC drivers. As of version 2.2.13, the unixODBC Driver Manager expects an 8-byte `SQLLEN` datatype. Starting with 15.7 ESD #4, Adaptive Server ODBC Driver installation contains both 4-byte `SQLLEN` and 8-byte `SQLLEN` versions of the driver. The 4-byte version is configured as the default. Please check the version of your unixODBC Driver Manager, and, if it is 2.2.13 or later, change your ODBC driver installation:

```
> cd ${SYBASE}/DataAccess64/ODBC/lib
> rm libsybdrvodb.so
> ln -s libsybdrvodb-sqllen8.so libsybdrvodb.so
```

Note that Red Hat Enterprise Linux version 6 and later use the 8-byte `SQLLEN` version of the unixODBC Driver Manager and thus require the aforementioned change.

## Building Applications Using an ODBC Driver Manager

Link ODBC applications to the Microsoft ODBC Driver Manager on Windows platform.

The Microsoft ODBC Driver Manager includes either a DLL named `odbc32.dll` or an import library named `odbc32.lib`

The `odbc32.dll` file is located in `%SystemRoot%\system32`. The `odbc32.lib` file can appear in a number of locations, depending on which products you have installed. If you use Microsoft Visual Studio.NET, the `odbc32.lib` is located in the `%Install Path to Microsoft Visual Studio%\ Vc7\PlatformSDK\Lib`.

To link an ODBC application to the Microsoft ODBC Driver Manager, use `odbc32.lib`.

### Using unixODBC Driver Manager

Build applications using the unixODBC Driver Manager on UNIX platform.

The unixODBC Driver Manager includes a shared library named `libodbc.so`, as a soft link to a library named `libodbc.so.1`. This file is typically located in the `/usr/lib` directory.

**Note:** Some older Driver Manager packages do not create the soft link from `libodbc.so.1` to `libodbc.so`. Sybase recommends that you manually create this link. The ODBC Driver Manager also includes another shared library called `libodbcinst.so.1`. A soft link from this file to `libodbcinst.so` should also exist. If it is not on your system, you should create one.

1. To link an ODBC application against the unixODBC Driver Manager, pass the `-lodbc` flag to the linker.
2. If the unixODBC Driver Manager is not installed in the `/usr/lib` directory, you must also pass this to the linker:

   ```
   -Ldir
   ```

   where `dir` is the directory where the unixODBC Driver Manager shared libraries are located.

### Using the Sybase iAnywhere ODBC Driver Manager

Build applications using the Sybase iAnywhere Driver Manager on UNIX platform.

1. To link an ODBC application against the Sybase iAnywhere ODBC Driver Manager, pass the `-lodbc` or `-ldbodm` flag to the linker.
2. You must also pass the `-L`*dir* flag to the linker.

   where *dir* is the directory where the Sybase iAnywhere ODBC Driver Manager shared libraries are located.

## Build Applications Without Using an ODBC Driver Manager

You can build applications without using an ODBC Driver Manager. The Adaptive Server ODBC Driver is a shared dynamic library with platform specific names.

| Platform | Library File | Location |
|---|---|---|
| Windows 32-bit | `sybdrvodb.dll` | `%SYBASE%\DataAccess\ODBC\dll` |
| Windows 64-bit | `syb-drvodb64.dl` | `%SYBASE%\DataAccess64\ODBC\dll` |
| UNIX 32-bit | `libsybdr-vodb.so` | `$SYBASE/DataAccess/ODBC/lib` |
| UNIX 64-bit | `libsybdr-vodb.so` | `$SYBASE/DataAccess64/ODBC/lib` |

### Linking an ODBC Application With the Adaptive Server ODBC Driver on Windows

Link an ODBC application on Microsoft Windows using the `sybdrvodb.lib` library file.

1. Add `sybdrvodb.lib` to `Additional Dependencies` in the `Linker/Input` properties and add *<aseodbc_dir>* to `Additional Library Directories` in `Linker/General` properties for your project.
2. When deploying your application, verify that `%SYBASE%\DataAccess\ODBC\dll` (for 32-bit ODBC drivers) or `%SYBASE%\DataAccess64\ODBC\dll` (for 64-bit ODBC drivers), the directory containing the Adaptive Server ODBC Driver shared library, is included in your system path.

### Linking an ODBC Application With the Adaptive Server ODBC Driver on UNIX

Link an ODBC application on UNIX using the `-lsybdrvodb` library file.

1. Pass the `-lsybdrvodb` and `-L`*<aseodbc_dir>* flags to the linker.
2. When deploying your application, verify that `$SYBASE/DataAccess/ODBC/lib` (for 32-bit ODBC drivers) or `$SYBASE/DataAccess64/ODBC/lib` (for 64-bit ODBC drivers), the directory containing the Adaptive Server ODBC Driver shared library, is included in your library path.

   The library path variable for your platform is:
   - On HP HP-UX Itanium – `SHLIB_PATH`
   - On IBM AIX – `LIBRARY_PATH`
   - On Linux and Solaris – `LD_LIBRARY_PATH`

## Adaptive Server ODBC Driver Samples

Review the Adaptive Server ODBC Driver samples for Windows and UNIX platforms.

The Adaptive Server ODBC Driver samples are located in:

- 32-bit Linux – `$SYBASE\DataAccess\ODBC\samples`
- 64-bit UNIX – `$SYBASE\DataAccess64\ODBC\samples`
- Microsoft Windows – `%SYBASE%\DataAccess\ODBC\samples` or `%SYBASE%\DataAccess64\ODBC\samples`

Each directory and sample includes a `README` file that contains instructions on building and running samples.

These samples are available on Microsoft Windows and UNIX:

- `advanced`

- `asynchexec`
- `cursors`
- `odbcbatch`
- `odbcloblocator`
- `simple`

These samples are available on Microsoft Windows only:

- `adovbsample`
- `kerberos`

## ODBC Handles

ODBC applications use a small set of handles to define basic features, such as database connections and SQL statements.

A handle is a 32-bit value on 32-bit platforms and a 64-bit value on 64-bit platforms. The handle types required for ODBC programs are:

| Item | Handle Type |
|---|---|
| **Environment** | `SQLHENV` |
| **Connection** | `SQLHDBC` |
| **Statement** | `SQLHSTMT` |
| **Descriptor** | `SQLHDESC` |

These handles are used in all ODBC applications:

- **Environment** – provides a global context to access data. Every ODBC application must allocate exactly one environment handle upon starting, and must free it at the end.
  This code allocates an environment handle:

```
SQLHENV env;
SQLRETURN rc;
rc = SQLAllocHandle( SQL_HANDLE_ENV,
       SQL_NULL_HANDLE, &env );
```

- **Connection** – is specified by an ODBC driver and a datasource. An application can have several connections associated with its environment. Allocating a connection handle does not establish a connection; a connection handle must be allocated first and then used when the connection is established.
  This code allocates a connection handle:

```
SQLHDBC  dbc;
SQLRETURN rc;
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

- **Statement** – provides access to a SQL statement and any information associated with it, such as result sets and parameters. Each connection can have several statements.

Statements are used for cursor operations (fetching data) and for single statement execution (such as **INSERT**, **UPDATE**, and **DELETE**).

This code allocates a statement handle:

```
SQLHSTMT stmt; SQLRETURN rc;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

- **Descriptor** – is a collection of metadata that describes the parameters of a SQL statement or the columns of a result set, as seen by the application or driver. Thus, a descriptor can fill any of four roles:
  - Application Parameter Descriptor (APD) – contains information about the application buffers bound to the parameters in an SQL statement, such as their addresses, lengths, and C datatypes.
  - Implementation Parameter Descriptor (IPD) – contains information about the parameters in a SQL statement, such as their SQL datatypes, lengths, and nullability.
  - Application Row Descriptor (ARD) – contains information about the application buffers bound to the columns in a result set, such as their addresses, lengths, and C datatypes.
  - Implementation Row Descriptor (IRD) – contains information about the columns in a result set, such as their SQL datatypes, lengths, and nullability.

  This example illustrates how to retrieve implicitly allocated descriptors:

```
SQLRETURN rc;
SQLHDESC aparamdesc;
SQLHDESC iparamdesc;
SQLHDESC irowdesc;
SQLHDESC arowdesc;
rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_PARAM_DESC,
   &aparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &arowdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &iparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &irowdesc, SQL_IS_POINTER);
```

  Implicit descriptors are automatically freed when the statement handle is freed by calling **SQLFreeHandle(SQL_HANDLE_STMT, stmt)**.

## Allocating an ODBC Handle

Use **SQLAllocHandle** and **SQLFreeHandle** functions to allocate ODBC handles.

1. Call the **SQLAllocHandle** function, which takes these parameters:
   - An identifier for the type of item being allocated
   - The handle of the parent item
   - A pointer to the location of the handle to be allocated

For a full description, see **SQLAllocHandle** in the *Microsoft ODBC Programmer's Reference*.

2. Use the handle in subsequent function calls.
3. Free the object using **SQLFreeHandle**, which takes these parameters:
   - An identifier for the type of item being freed
   - The handle of the item being freed

   For a full description, see **SQLFreeHandle** in the *Microsoft ODBC Programmer's Reference*.

   For example, this code fragment allocates and frees an environment handle:

```
SQLHENV env;
SQLRETURN retcode;
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if ( retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO )
{
// success: application code here
}
retcode = SQLFreeHandle(SQL_HANDLE_ENV, env);
```

# Connections to a Datasource

Use ODBC functions to establish a connection to an Adaptive Server Enterprise database.

**Note:** In general, the examples use **SQLConnect**.

## Choose an ODBC Connection Function

ODBC supplies a set of connection functions.

Which of these functions you use depends on how you expect your application to be deployed and used:

- **SQLConnect** – the simplest connection function

  **SQLConnect** – takes a datasource name (DSN), and an optional user ID and password. You might want to use **SQLConnect** if you hard-code a datasource name into your application.

  For more information, see **SQLConnect** in the *Microsoft ODBC Programmer's Reference*.
- **SQLDriverConnect** – connects to a datasource using a connection string

  **SQLDriverConnect** – allows the application to use Adaptive Server Enterprise-specific connection information that is external to the datasource.

  **Note:** On UNIX, the Adaptive Server ODBC Driver supports only **SQL_DRIVER_NOPROMPT**.

  You can also use **SQLDriverConnect** to connect without specifying a datasource.

For more information, see **SQLDriverConnect** in the *Microsoft ODBC Programmer's Reference*.

- **SQLBrowseConnect** – connects to a datasource using a connection string, like **SQLDriverConnect**.

  **SQLBrowseConnect** – allows your application to build its own dialog boxes to prompt for connection information, and to browse for datasources used by a particular driver in this case, the Adaptive Server ODBC Driver.

  For more information, see **SQLBrowseConnect** in the *Microsoft ODBC Programmer's Reference*.

### See also

- *Connections to a Database* on page 27

## Establishing an ODBC Connection

Establish a connection for your application, before it can carry out any database operations.

1. Allocate an ODBC environment:

```
SQLHENV env;
SQLRETURN retcode;
retcode = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

2. Declare the ODBC version.

   By declaring that the application follows ODBC version 3, SQLSTATE values and some other version-dependent features are set to the proper behavior.

   For example:

```
retcode = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
   (void*)SQL_OV_ODBC3, 0);
```

3. If necessary, assemble the datasource or connection string.

   Depending on your application, you can have a hard-coded datasource or connection string, or you can store it externally for greater flexibility.

4. Allocate an ODBC connection handle:

```
retcode = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

5. Before connecting, set any connection attributes that are required.

   Some connection attributes must be set before establishing a connection, while others can be set either before or after.

   For example:

```
retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_IS_UINTEGER);
```

6. Call the ODBC connection function:

```
if (retcode == SQL_SUCCESS || retcode ==
   SQL_SUCCESS_WITH_INFO)
```

```
{
   printf( "dbc allocated\n" );
   retcode = SQLConnect( dbc, (SQLCHAR*) "MANGO",
      SQL_NTS, (SQLCHAR*) "sa", SQL_NTS,
      (SQLCHAR*) "", SQL_NTS );
   if (retcode == SQL_SUCCESS || retcode ==
      SQL_SUCCESS_WITH_INFO)
      {
         // successfully connected.
      }
}
```

You can find a complete sample of establishing a connection in your installation directory.

Notes on Usage

- Every string passed to ODBC has a corresponding length. If the length is unknown, you can pass **SQL_NTS** indicating that it is a Null Terminated String whose end is marked by the null character (\0).
- Use the **SQLSetConnectAttr** function to control details of the connection. For example, this statement turns off ODBC **autocommit** behavior:

```
retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_IS_UINTEGER );
```

Many aspects of the connection can be controlled through the connection parameters.

For more information including a list of connection attributes, see **SQLSetConnectAttr** in the *Microsoft ODBC Programmer's Reference*.

### See also

- *Connections to a Database* on page 27

## Threads and Connections in ODBC Applications

You can develop multithreaded ODBC applications for Adaptive Server Enterprise. Sybase recommends that you use a separate connection for each thread. However, you are allowed to share an open connection among multiple threads.

# SQL Statements Execution

ODBC includes several functions for executing SQL statements.

- **Direct execution** – Adaptive Server parses the SQL statement, prepares an access plan, and executes the statement. Parsing and access plan preparation are called preparing the statement.
- **Prepared execution** – the statement preparation is carried out separately from the execution. For statements that are to be executed repeatedly, this avoids repeated preparation and as a result improves performance.

- **Bound parameter execution** – you can construct and execute a SQL statement using bound parameters to set values for statement parameters at runtime. Bound parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once.

## Executing a SQL Statement in an ODBC Application

Use the **SQLExecDirect** function to prepare and execute a SQL statement.

Optionally, the statement can include parameters. The **SQLExecDirect** function takes a statement handle, a SQL string, and a length or termination indicator, which in this example below is a null-terminated string indicator.

This code fragment illustrates how to execute a statement without parameters.

1. Allocate a handle for the statement using **SQLAllocHandle**.

   For example, this statement allocates a **SQL_HANDLE_STMT** handle with the name "stmt", on a connection with a handle named "dbc":

   ```
   SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
   ```

2. Call the **SQLExecDirect** function to execute the statement.

   For example, these lines declare a statement and execute it:

   ```
   SQLCHAR *deletestmt =
      "DELETE FROM department WHERE dept_id = 201";
   SQLExecDirect( stmt, deletestmt, SQL_NTS) ;
   ```

See **SQLExecDirect** in the *Microsoft ODBC Programmer's Reference*.

## Executing SQL Statements With Bound Parameters

Construct and execute a SQL statement, using bound parameters to set values for statement parameters at runtime.

1. Allocate a handle for the statement using **SQLAllocHandle**.

   For example, this statement allocates a **SQL_HANDLE_STMT** handle the with name "stmt", on a connection with a handle named "dbc":

   ```
   SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
   ```

2. Bound parameters for the statement using **SQLBindParameter**.

   For example, these lines declare variables to hold the values for the department ID, department name, and manager ID, as well as for the statement string itself. Then, they bind parameters to the first, second, and third parameters of a statement executed using the "stmt" statement handle:

   ```
   #defined DEPT_NAME_LEN 20

   SQLLEN cbDeptID = 0,
      cbDeptName = SQL_NTS, cbManagerID = 0;
   ```

```
SQLCHAR deptname[ DEPT_NAME_LEN ];
SQLSMALLINT deptID, managerID;
SQLCHAR *insertstmt =
   "INSERT INTO department "
   "( dept_id, dept_name, dept_head_id )"
   "VALUES (?, ?, ?,)";
SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,
   SQL_C_SSHORT, SQL_INTEGER, 0, 0,
   &deptID, 0, &cbDeptID);
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,
   deptname, 0,&cbDeptName);
SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,
   SQL_C_SSHORT, SQL_INTEGER, 0, 0,
   &managerID, 0, &cbManagerID);
```

3. Assign values to the parameters.

   For example, these lines assign values to the parameters for the fragment of step 2:

   ```
   deptID = 201;
   strcpy( (char * ) deptname, "Sales East" );
   managerID = 902;
   ```

   Usually, these variables are set in response to user action.

4. Execute the statement using **SQLExecDirect**.

   For example, this line executes the statement string held in “insertstmt” on the “stmt” statement handle:

   ```
   SQLExecDirect( stmt, insertstmt, SQL_NTS) ;
   ```

Bind parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once.

See **SQLExecDirect** in the *Microsoft ODBC Programmer's Reference*.

## Executing Prepared SQL Statements

Use the full set of functions that comes with Adaptive Server ODBC Driver for using prepared statements, which provide performance advantages for statements that are used repeatedly.

1. Prepare the statement using **SQLPrepare**.

   For example, this code fragment illustrates how to prepare an **insert** statement:

   ```
   SQLRETURN retcode;
   SQLHSTMT stmt;
   retcode = SQLPrepare( stmt, "INSERT INTO department"
      "( dept_id, dept_name, dept_head_id )"
      "VALUES (?, ?, ?,)", SQL_NTS);
   ```

   where:

- *retcode* – holds a return code that should be tested for success or failure of the operation.
- *stmt* – provides a handle to the statement.
- ? – is a statement parameter marker.

2. Set statement parameter values using **SQLBindParameter**.

   For example, this function call sets the value of the *dept_id* variable:

```
SQLBindParameter( stmt,
   1,
   SQL_PARAM_INPUT,
   SQL_C_SHORT,
   SQL_INTEGER,
   0,
   0,
   &sDeptID,
   0,
   &cbDeptID);
```

   where:

   - *stmt* – is the statement handle.
   - 1 – indicates that this call sets the value of the first parameter.
   - *SQL_PARAM_INPUT* – indicates that the parameter is an input statement.
   - *SQL_C_SHORT* – indicates the C datatype being used in the application.
   - *SQL_INTEGER* – indicates the SQL datatype being used in the database.
   - 0 – indicates the column precision.
   - 0 – indicates the number of decimal digits.
   - *&sDeptID* – is a pointer to a buffer for the parameter value.
   - 0 – indicates the length of the buffer, in bytes.
   - *&cbDeptID* – is a pointer to a buffer for the length of the parameter value.

3. Bind the other two parameters and assign values to **sDeptId**:

```
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,
   deptname, 0,&cbDeptName);

SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,
   SQL_C_SSHORT, SQL_INTEGER, 0, 0,
   &managerID, 0, &cbManagerID);
```

4. Execute:

```
retcode = SQLExecute( stmt);
```

   You can repeat steps 2 through 4 multiple times.

5. Drop the statement using **SQLFreeHandle**.

   Dropping the statement frees resources associated with the statement itself.

# Result Sets

ODBC applications use cursors to manipulate and update result sets. The Adaptive Server ODBC Driver provides extensive support for different kinds of cursors and cursor operations.

## Choosing Cursor Characteristics

Use cursors to carry out tasks for ODBC functions that execute statements and manipulate result sets.

Applications open a cursor implicitly when they execute a statement that returns a result set.

For applications that move through a result set only in a forward direction and do not update the result set, cursor behavior is relatively straightforward. By default, ODBC applications request this behavior. ODBC defines a read-only, forward-only cursor, and the Adaptive Server ODBC Driver provides a cursor optimized for performance in this case.

To set the required ODBC cursor characteristics, call the **SQLSetStmtAttr** function that defines statement attributes. Call **SQLSetStmtAttr**, before executing a statement that returns a result set.

You can use **SQLSetStmtAttr** to set cursor characteristics. The characteristic that determines the cursor type for the Adaptive Server ODBC Driver is **SQL_ATTR_CONCURRENCY**.

Set one of these values:

- **SQL_CONCUR_READ_ONLY** - disallows updates. This is the default.
- **SQL_CONCUR_LOCK**- uses the lowest level of locking needed to verify that the row is updated.

See **SQLSetStmtAttr** in the *Microsoft ODBC Programmer's Reference*.

For example, this fragment requests an updateable cursor:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY,
   SQL_CONCUR_LOCK, 0 );
```

## UseCursor Connection Property

The `UseCursor` connection property determines the type of cursor generated by the Adaptive Server ODBC Driver.

Adaptive Server ODBC Driver creates either of the two types of cursors, when a SQL statement that generate result sets is executed:

- Server-side cursors – uses more resources, but is required to fully support cursor semantics.
- Client-side cursors – uses less resources and is adequate for most use cases.

Valid values are:

- 0 – (default) Client-side cursors are used for all statements that generate result sets.
- 1 – Server-side cursors are used for all statements that generate result sets.
- 2 – Server-side cursors are used for all statements that generate result sets only when the **SQLSetCursorName** ODBC function is called. Use this setting to limit use of server-side cursors only for situations that require them.

**Note:** Depending on the other cursor attribute settings, a server-side cursor request may implicitly be changed by the Adaptive Server ODBC Driver to a client-side cursor.

## Retrieving Data

Retrieve data from a database using SQL statements.

1. Execute a **select** statement using **SQLExecute** or **SQLExecDirect** to retrieve rows from a database.

   This opens a cursor on the statement.
2. Use **SQLFetch** or **SQLFetchScroll** with **SQL_FETCH_NEXT** option to fetch rows through the cursor.

   When an application frees the statement using **SQLFreeStmt** with **SQL_CLOSE** option, it closes the cursor.
3. Fetch values from a cursor using either **SQLBindCol** or **SQLGetData**:
   - If you use **SQLBindCol**, values are automatically retrieved on each fetch.
   - If you use **SQLGetData**, you must call it for each column after each fetch.

   **SQLGetData** is used to fetch values in pieces for columns such as `LONG VARCHAR` or `LONG BINARY`.
4. Optionally, set the **SQL_ATTR_MAX_LENGTH** statement attribute to a value large enough to hold the entire value for the column.

   For **SQL_ATTR_MAX_LENGTH**, the default value is 32KB.

   This code fragment from the `simple` sample opens a cursor on a query and retrieves data through the cursor. Error checking has been omitted to make the example easier to read.

```
SQLExecDirect( stmt, "select au_fname from authors",
   SQL_NTS ) ;
retcode = SQLBindCol( stmt, 1, SQL_C_CHAR, aufName,
   sizeof(aufName), &aufNameLen);
while(retcode == SQL_SUCCESS || retcode ==
   SQL_SUCCESS_WITH_INFO)
{
   retcode = SQLFetch( stmt );
}
```

## Updating and Deleting Rows Through a Cursor

Set the statement attribute **SQL_ATTR_CONCURRENCY** to **SQL_CONCUR_LOCK** to open a cursor to update or delete rows.

```
SQLSetStmtAttr(stmt,SQL_ATTR_CONCURRENCY,(SQLPOINTER)
   SQL_CONCUR_LOCK,0);
```

This code fragment from the `cursor` sample illustrates using cursors for updates and deletes. Error checking has been omitted for clarity.

```
/* Set statement attribute for an updateable cursor */
SQLSetStmtAttr(stmt, SQL_ATTR_CONCURRENCY,
   (SQLPOINTER)SQL_CONCUR_LOCK, 0);
SQLSetCursorName(stmt1, "CustUpdate", SQL_NTS);
SQLExecDirect(stmt1, "select LastName
   from t_CursorTable ", SQL_NTS) ;
SQLFetch(stmt1);
SQLExecDirect(stmt2, "Update t_CursorTable"
   "set LastName='UpdateLastName'"
   "where current of CustUpdate", SQL_NTS) ;
```

For the complete code, refer to the `cursor.cpp` sample.

## Scrollable Cursors

Scrollable cursors can go backward as well as forward to more easily support screen-based applications.

When a user scrolls backward and forward, the backend provides the corresponding data.

### Setting the UseCursor Connection Property

Set the `UseCursor` connection property to determine whether the client-side or the server-side scrollable cursors are used.

- When the `UseCursor` connection property is 1 or 2, server-side scrollable cursors are used if Adaptive Server version is 15.0 or later.

  In earlier versions of the Adaptive Server, server-side scrollable cursors were not available.
- When the `UseCursor` connection property is 0, client-side scrollable cursors (cached result sets) are used, regardless of the Adaptive Server version.

**Warning!** Using client-side scrollable cursors is resource-intensive.

#### See also

- *UseCursor Connection Property* on page 14

### Static Insensitive Scrollable Cursor Support

The Adaptive Server ODBC Driver supports the `Static Insensitive` scrollable cursor. The `Static Insensitive` implements the ODBC `SQLFetchScroll` method to **scroll** and **fetch** rows.

The `SQLFetchScroll` method is a standard ODBC method defined in *Microsoft Open Database Connectivity Software Development Kit Programmer's Reference, Volume 2*, which is part of the MSDN library.

The Adaptive Server ODBC Driver supports these scrolling types:

- `SQL_FETCH_NEXT` – returns the next rowset.
- `SQL_FETCH_PRIOR` – returns the prior rowset.
- `SQL_FETCH_RELATIVE` – returns the rowset *n* from the start of the current rowset.
- `SQL_FETCH_FIRST` – returns the first rowset in the result set.
- `SQL_FETCH_LAST` – returns the last complete rowset in the result set.
- `SQL_FETCH_ABSOLUTE` – returns the rowset starting at row *n*.

### Scrollable Cursor Attributes

The scrollable cursor attirbutes in the Adaptive Server ODBC Driver.

You must set these attributes to use scrollable cursors:

- **SQL_ATTR_CURSOR_SCROLLABLE** – the type of scrollable cursor you are using. It should be set to the value of **SQL_SCROLLABLE**. Possible values for **SQL_ATTR_CURSOR_SENSITIVITY** are `static`, `semi-sensitive`, and `insensitive`.
- **SQL_ATTR_CURSOR_SENSITIVITY** – the sensitivity value for this scrollable cursor. The only supported value for this is `SQL_INSENSITIVE`.

These are optional attributes when using scrollable cursors:

- **SQL_ATTR_ROW_ARRAY_SIZE** – the number of rows that you want returned from each call to the `SQLFetchScroll()` method. If you do not set this value, the default value of one row is used.
- **SQL_ATTR_CURSOR_TYPE** – the type of scrollable cursor you are using. The only supported values for this are `SQL_CURSOR_FORWARD_ONLY` or `SQL_CURSOR_STATIC`.
- **SQL_ATTR_ROWS_FETCHED_PTR** – the address where the number of rows fetched are stored. The **SQL_ATTR_ROWS_FETCHED_PTR** points to a variable of datatype `SQLUINTEGER`.
- **SQL_ATTR_ROW_STATUS_PTR** – the address where the row status is stored. The **SQL_ATTR_ROW_STATUS_PTR** points to a variable of datatype `SQLUSMALLINT`.

### Executing Scrollable Cursors

Use `SQLBindCol()` and `SQLFetchScroll()` APIs to execute a scrollable cursor.

1. Set the scrollable cursor attributes for your environment.
2. Bind the results.

   For example, add this code to your program:

   ```
   res=SQLBindCol(m_StatementHandle, 2, SQL_C_DOUBLE, price, 0,
   NULL);
   ```

   ```
   res=SQLBindCol(m_StatementHandle, 3, SQL_C_LONG, quantity, 0,
   NULL);
   ```

3. Scroll and fetch by using `SQLFetchScroll()`.

   For example, add this code to your program:

   ```
   res = SQLSetStmtAttr(m_StatementHandle,
   SQL_ATTR_CURSOR_SCROLLABLE,
      (SQLPOINTER)SQL_SCROLLABLE,SQL_IS_INTEGER);

   res = SQLSetStmtAttr(m_StatementHandle,
   SQL_ATTR_CURSOR_SENSITIVITY,
      (SQLPOINTER)SQL_INSENSITIVE, SQL_IS_INTEGER);
   ```

4. Execute the **Select** statement.

   For example, add this code to your program:

   ```
   res = SQLExecDirect(m_StatementHandle,
      (SQLCHAR "select price, quantity from book" SQL_NTS);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_NEXT,0);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_PRIOR,0);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_FIRST,0);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_LAST,0);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,2);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,-2);
   res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_RELATIVE,1);
   ```

5. Close the result set and the cursor.

   For example, add this code to your program:

   ```
   res = SQLFreeStmt(m_StatementHandle,SQL_CLOSE);
   ```

### See also

- *Scrollable Cursor Attributes* on page 17

### View Results

View results after executing a scrollable cursor, assuming a total of *N* rows and a rowset *m* where *N* > *m*.

| Result | Interpretation |
|---|---|
| Absolute 0 | No row is returned, error. |
| Absolute 1 | *m* row is returned. |
| Absolute *N* | 1 row is returned. |
| Absolute *N*+1 | No row is returned, error. |
| First | The first (1..*m* ) rows are returned. |
| Last | The last (*N*-*m*+1 .. *N*) rows are returned. |
| Next | The same as `SQLFetch()`. |
| Prior | Return the rowset that is before current rowset. |

These results are expected if the current cursor points to row *k* and *k*-*a* > 0, *k* + *m* + *a* < *N*, *a*>=0:

| Result | Interpretation |
|---|---|
| Relative -*a* | The rows (*k*-*a*, *k*-*a* + *m* -1) are returned. |
| Relative *a* | The rows (*k* + *a*, *k*+*a* + *m* -1) are returned. |

### Scrollable Cursor Attributes Set Implicitly

Certain attributes are set implicitly when your application sets specific attributes.

The supported ODBC scrollable cursor attributes set implicitly are:

| Application Sets Attribute to | Other Attributes Set Implicitly |
|---|---|
| SQL_ATTR_CONCURRENCY to SQL_CONCUR_READ_ONLY | SQL_ATTR_CURSOR_SENSITIVITY to SQL_INSENSITIVE |
| SQL_ATTR_CONCURRENCY to SQL_CONCUR_LOCK | SQL_ATTR_CURSOR_SENSITIVITY to SQL_SENSITIVE |
| SQL_ATTR_CURSOR_SCROLLABLE to SQL_NONSCROLLABLE | SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_FORWARD_ONLY |
| SQL_ATTR_CURSOR_SENSITIVITY to SQL_INSENSITIVE | SQL_ATTR_CONCURRENCY to SQL_CONCUR_READ_ONLY SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_STATIC |

| Application Sets Attribute to | Other Attributes Set Implicitly |
|---|---|
| SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_FORWARD_ONLY | SQL_ATTR_CURSOR_SCROLLABLE to SQL_NONSCROLLABLE |
| SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_STATIC | SQL_ATTR_CURSOR_SCROLLABLE to SQL_SCROLLABLE |

## Calling Stored Procedures

Create and call stored procedures to process the results from an ODBC application.

For a full description of stored procedures and triggers, see the *Adaptive Server Enterprise Reference Manual*.

There are two types of procedures, those that return result sets, and those that do not.

You can use **SQLNumResultCols** to tell the difference: The number of result columns is zero if the procedure does not return a result set. If there is a result set, you can fetch the values using **SQLFetch** or **SQLFetchScroll** just like any other cursor.

1. Pass parameters to procedures using parameter markers (question marks).
2. Use **SQLBindParameter** to assign a storage area for each parameter marker, whether it is an **INPUT**, **OUTPUT**, or **INOUT** parameter.

   For example:

   The `advanced` sample illustrates a stored procedure that returns an output parameter and a return value, and another stored procedure that returns multiple result sets. Error checking has been omitted to make the example easier to read.

```
/*
Example 1: How to call a stored procedure and use input and output
parameters*/

SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &retVal, 0, SQL_NULL_HANDLE);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 4, 0, stor_id, sizeof(stor_id), SQL_NULL_HANDLE);
SQLBindParameter(stmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR,
   SQL_VARCHAR, 20, 0, ord_num, sizeof(ord_num), &ordnumLen);
SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_VARCHAR, 40, 0, date, sizeof(date), &dateLen);

SQLExecDirect( stmt, "{ ? = call sp_selectsales(?,?,?) }",
SQL_NTS);
/*
At this point retVal contains the return value as returned from
the stored
procedure and the ord_num contains the order number as returned
from the stored procedure
*/
```

```
/*
Example 2: How to call stored procedures returning multiple result
sets
*/

SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR , 4, 0, stor_id, sizeof(stor_id), SQL_NULL_HANDLE);

SQLExecDirect(stmt, "{ call sp_multipleresults(?) }", SQL_NTS);
SQLBindCol( stmt, 1, SQL_C_CHAR, dbValue, sizeof(dbValue),
&dbValueLen);
SQLSMALLINT count = 1;

while(retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
   retcode = SQLFetch( stmt );
   if (retcode == SQL_NO_DATA)
   {
     /*
     -- End of first result set --
     */
     if(count == 1)
     {
        retcode = SQLMoreResults(stmt);
        count ++;
     }
     /*
      At this point dbValue contains the value in the current row
of the
      result
     */
   }
}
```

## Error Handling

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the **SQLGetDiagFieId** function or the **SQLGetDiagRec** function.

The **SQLError** function was used in ODBC versions up to, but not including, version 3. As of version 3, the **SQLError** function has been replaced by the **SQLGetDiagRec** and **SQLGetDiagField** functions.

Every ODBC function returns a **SQLRETURN** that is one of these status codes:

| Status Code | Description |
|---|---|
| SQL_SUCCESS | No error. |
| SQL_ERROR | An error occured. A call to **SQLGetDiagRec** will retrieve the message. |

| Status Code | Description |
|---|---|
| SQL_SUCCESS_WITH_INFO | The function completed, but a call to **SQLGetDiagRec** will indicate a warning.<br>The most common cause for this status is that a value being returned is too long for the buffer provided by the application. |
| SQL_INVALID_HANDLE | An invalid environment, connection, or statement handle was passed as a parameter.<br>This often happens if a handle is used after it has been freed, or if the handle is the null pointer. |
| SQL_NO_DATA | There is no information available.<br>The most common use for this status is when fetching from a cursor; it indicates that there are no more rows in the cursor. |
| SQL_NEED_DATA | Data is needed for a parameter.<br>This is an advanced feature described in the ODBC Software Development Kit documentation under **SQLParamData** and **SQLPutData**. |

Every environment, connection, and statement handle can have one or more errors or warnings associated with it. Each call to **SQLGetDiagRec** returns the information for one error and removes the information for that error. If you do not call **SQLGetDiagRec** to remove all errors, the errors are removed on the next function call that passes the same handle as a parameter. The exception to this is subsequent calls to SQLExecute during a batch.

Each call to **SQLGetDiagRec** can pass either an environment, connection, or statement handle. The first call passes in a handle of type **SQL_HANDLE_DBC** to get the error associated with a connection. The second call passes in a handle of type **SQL_HANDLE_STMT** to get the error associated with the statement that was just executed.

**SQLGetDiagRec** returns SQL_SUCCESS if there is an error to report (not SQL_ERROR), and SQL_NO_DATA_FOUND if there are no more errors to report.

These code fragments use **SQLGetDiagRec** and return codes:

```
retcode = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt );
if( retcode == SQL_ERROR )
{
   SQLGetDiagRec(SQL_HANDLE_DBC,dbc, 1, NULL, NULL,
     errmsg, 100, NULL);
   /* Assume that print_error is defined */
   print_error( "Allocation failed", errmsg );
   return;
}
```

```
retcode = SQLExecDirect( stmt,
   "delete from sales_order_items where id=2015",
```

```
   SQL_NTS );
if( retcode == SQL_ERROR )
{
   SQLGetDiagRec(SQL_HANDLE_STMT,stmt, 1, NULL, NULL,
      errmsg, 100, NULL);
   /* Assume that print_error is defined */
   print_error( "Failed to delete items", errmsg );
   return;
}
```

## Datatype Mappings

The Adaptive Server ODBC Driver datatype mappings.

| Adaptive Server Datatype | ODBC SQL Datatype | ODBC Bind Datatype |
|---|---|---|
| bigdatetime | SQL_TYPE_TIMESTAMP | SQL_C_TYPE_TIMESTAMP or SQL_C_CHAR |
| bigtime | SQL_TYPE_TIME | SQL_C_TYPE_TIME or SQL_C_CHAR |
| bigint | SQL_BIGINT | SQL_C_BIGINT |
| binary | SQL_BINARY | SQL_C_BINARY |
| bit | SQL_BIT | SQL_C_BIT |
| char | SQL_CHAR | SQL_C_CHAR |
| date | SQL_TYPE_DATE | SQL_C_TYPE_DATE or SQL_C_CHAR |
| datetime | SQL_TYPE_TIMESTAMP | SQL_C_TYPE_TIMESTAMP or SQL_C_CHAR |
| decimal | SQL_DECIMAL | SQL_C_NUMERIC or SQL_C_CHAR or SQL_C_PACKEDBCD |
| double | SQL_DOUBLE | SQL_C_DOUBLE |
| float(<16) | SQL_REAL | SQL_C_FLOAT |
| float(>=16) | SQL_DOUBLE | SQL_C_DOUBLE |
| image | SQL_LONGVARBINARY | SQL_C_BINARY |
| image_locator | SQL_IMAGE_LOCATOR | SQL_C_ IMAGE_LOCATOR |

| Adaptive Server Datatype | ODBC SQL Datatype | ODBC Bind Datatype |
|---|---|---|
| integer | SQL_INTEGER | SQL_C_LONG |
| money | SQL_DECIMAL | SQL_C_NUMERIC or SQL_C_CHAR or SQL_C_PACKEDBCD |
| nchar | SQL_CHAR | SQL_C_CHAR |
| nvarchar | SQL_VARCHAR | SQL_C_CHAR |
| numeric | SQL_NUMERIC | SQL_C_NUMERIC or SQL_C_CHAR or SQL_C_PACKEDBCD |
| real | SQL_REAL | SQL_C_FLOAT |
| smalldatetime | SQL_TYPE_TIMESTAMP | SQL_C_TYPE_TIMESTAMP or SQL_C_CHAR |
| smallint | SQL_SMALLINT | SQL_C_SHORT |
| smallmoney | SQL_DECIMAL | SQL_C_NUMERIC or SQL_C_CHAR or SQL_C_PACKEDBCD |
| text | SQL_LONGVARCHAR | SQL_C_CHAR |
| text_locator | SQL_TEXT_LOCATOR | SQL_C_TEXT_LOCATOR |
| time | SQL_TYPE_TIME | SQL_C_TYPE_TIME or SQL_C_CHAR |
| timestamp | SQL_BINARY | SQL_C_BINARY |
| tinyint | SQL_TINYINT | SQL_C_TINYINT |
| unichar | SQL_WCHAR | SQL_C_CHAR |
| unitext | SQL_WLONGVARCHAR | SQL_C_CHAR |
| unitext_locator | SQL_UNITEXT_LOCATOR | SQL_C_ UNITEXT_LOCATOR |
| univarchar | SQL_WVARCHAR | SQL_C_CHAR |
| unsignedbigint | SQL_BIGINT | SQL_C_UBIGINT |
| unsignedint | SQL_INTEGER | SQL_C_ULONG |

| Adaptive Server Datatype | ODBC SQL Datatype | ODBC Bind Datatype |
|---|---|---|
| unsigneds-mallint | SQL_SMALLINT | SQL_C_USHORT |
| varbinary | SQL_VARBINARY | SQL_C_BINARY |
| varchar | SQL_VARCHAR | SQL_C_CHAR |

### *Special Instructions for Unichar, Varchar, and Unitext*

When you use the Adaptive Server datatypes unichar, univarchar, and unitext, and then bind any of them to SQL_C_CHAR, the Adaptive Server ODBC Driver must convert the data from Unicode to multibyte and vice versa. For this conversion, it must have the SYBASE charsets installed in the $SYBASE directory. The installation program includes an option to install these charset files. The charsets are required if char, varchar and unitext are used and the client charset is different than the server.

If the driver does not find the charsets, or if the $SYBASE environment variable is not set, then an appropriate error is propagated to the application. To install the SYBASE charsets, you must reinstall the ODBC Driver. See the *Software Developer's Kit and Open Server Installation Guide* for your platform.

**Note:** To support older applications, the Adaptive Server ODBC Driver assumes that the default type is SQL_C_CHAR when a unitext, univarchar, or unichar column is bound as SQL_C_DEFAULT. To bind as unicode, the application must explicitly use a bind type of SQL_C_WCHAR.

### *Special Instructions for Bigint*

When you use a column of type bigint as an identifier in an Adaptive Server table (for example, as an identity or primary key), and applications such as Microsoft Access accesses the table through Adaptive Server ODBC Driver, the values of such column may appear as "#deleted", and prevent further operations on the table. As a workaround, set CHANGEBIGINTDEFAULT to 1.

CHANGEBIGINTDEFAULT values:

- 0 – the default value, binds SQL_C_DEFAULT to SQL_C_BIGINT.
- 1 – binds SQL_C_DEFAULT to SQL_C_CHAR. Use this setting when you want to access Adaptive Server tables with bigint identifiers from applications such as Microsoft Access or Microsoft Excel.
- 2 – binds SQL_C_DEFAULT to SQL_C_WCHAR.

# Connections to a Database

Any client application that uses Adaptive Server Enterprise must establish a connection to the Adaptive server before any work can be done.

The connection forms a channel through which all activity from the client application takes place. For example, your user ID determines permissions to carry out actions on the database and the database server has your user ID because it is part of the request to establish a connection. The Adaptive Server ODBC Driver uses connection information included in the call from the client application (perhaps together with information held on disk in an initialization file) to locate and connect to an Adaptive Server server running the required database.

## Connection Attributes

In version 15.7 ESD #7, Adaptive Server ODBC Driver adds support for setting client connection attributes efficiently using the ODBC `SQLSetConnectAttr` API.

The attribute values set are visible in the Adaptive Server `sysprocesses` table and help distinguish different client connections.

To set these attributes in versions earlier than 15.7 ESD #7, application programs had to explicitly call **set** statements to set corresponding attributes resulting in additional executions on the server. When the `SQLSetConnectAttr` API is used, the driver defers executing the **set** statements, attaching them to the next statement that is executed.

**Note:** Since the attribute values are not sent immediately after `SQLSetConnectAttr` API is called, the values set are invisible on Adaptive Server until the next statement is executed.

`SQLSetConnectAttr` supports these attributes:

- **SQL_ATTR_CLIENT_NAME** – sets the client name, it is equivalent to using the command set *clientname <value>*.
- **SQL_ATTR_CLIENT_HOST_NAME** – sets the client host name, it is equivalent to using the command set *clienthostname <value>.*
- **SQL_ATTR_CLIENT_APPL_NAME** – sets the client application name, it is equivalent to using the command set *clientapplname <value>*.

The value of these attributes is truncated to 30 bytes. Use the ODBC `SQLGetConnectAttr` to retrieve the value of these attributes. However, it does not reflect any changes to the server value made outside of this interface.

# Installing ODBC MetaData Stored Procedures

Install ODBC metadata stored procedures to ensure that ODBC functionalities behave as expected. Sybase recommends that you check the version of the ODBC MetaData stored procedures on all the Adaptive Server servers that you need to connect to, using the ODBC Driver and update them wherever needed.

### Prerequisites

Verify that you have permission to create stored procedures in `sybsystemprocs` to run the script.

**Note:** Use the **ODBC driver version information** utility with the `-connect` option to check if the metdata stored procedures are up to date or need update.

### Task

1. Go to the `sp` directory under the Adaptive Server ODBC Driver installation directory:
   - Adaptive Server ODBC Driver 32-bit for Microsoft Windows: `%SYBASE%\DataAccess\ODBC\sp`
   - Adaptive Server ODBC Driver 64-bit for Microsoft Windows: `%SYBASE%\DataAccess64\ODBC\sp`
   - Adaptive Server ODBC Driver 32-bit for Linux: `$SYBASE\DataAccess\ODBC\sp`
   - Adaptive Server ODBC Driver 64-bit for UNIX: `$SYBASE\DataAccess64\ODBC\sp`
2. Execute the `install_odbc_sprocs` script.
   - Adaptive Server ODBC Driver for Microsoft Windows:

     ```
     install_odbc_sprocs ServerName username
        [password]
     ```
   - Adaptive Server ODBC Driver for UNIX:

     ```
     ./install_odbc_sprocs ServerName username
        [password]
     ```

   where:
   - *ServerName* is the name of the Adaptive Server.
   - *username* is the user name to connect to the server.
   - *[password]* is the password for the user name. If the value is null, leave the parameter empty.

**See also**

- *ODBC Driver Version Information Utility* on page 46

## How Connection Parameters Work

When an application connects to a database, it uses a set of connection parameters to define the connection.

Connection parameters include information such as the server name, the database name, and a user ID. A keyword-value pair (of the form `parameter=value`) specifies each connection parameter. For example, you specify the user ID connection parameter as:

```
UID=sa
```

### *Connection Parameters Passed as Connection Strings*

Connection parameters are passed to the Adaptive Server ODBC driver as a connection string and are separated by semicolons:

```
parameter1=value1;parameter2=value2;...
```

In general, the connection string built by an application and passed to the driver does not correspond directly to the way a user enters the information. Instead, a user can fill in a dialog box, or the application can read connection information from an initialization file.

## Character Sets

The `CharSet` connection property defines the character set that the driver uses to send character data to Adaptive Server, while the `ClientCharset` connection property defines the character set used by client applications.

The valid `CharSet` values are:

- ServerDefault – when specified, Adaptive Server ODBC Driver communicates with Adaptive Server using the server's default character set. The Adaptive Server ODBC Driver converts character data for the client if the client and server use different character sets.
- ClientDefault – when specified, Adaptive Server ODBC Driver communicates with Adaptive Server using the client-specified character set. In this case, if the default Adaptive Server character set is different from the client's, Adaptive Server converts character data to the client character set. Adaptive Server uses additional resources when performing character set conversions.
- NoConversions – when specified, Adaptive Server ODBC Driver ignores the client's character set and does not convert character data. In this setting, the client application must ensure that character data is correctly converted between the client's character set and the default Adaptive Server character set. Use this value only under specific circumstances.

For example, when character data stored in Adaptive Server must be converted in the client application using a customized character set conversion logic.

**Note:** In the Microsoft Windows ODBC Data Source Administrator, the **Server Default**, **Client Charset**, and **No Conversions** fields found in the Advanced window correspond to the `CharSet` values ServerDefault, ClientDefault, and NoConversions, respectively.

The Adaptive Server ODBC Driver determines the client character set, depending on the platform:

- On Microsoft Windows, the default client character set selected is the ANSI code page for your login session. The valid code page types are ANSI, OEM, and Other. If Other is chosen, you must enter a valid Windows code page value.
- By default, on UNIX, the Adaptive Server ODBC Driver examines the LC_CTYPE and LANG environment variables. If they are not set, the driver defaults to ISO 8859-1. If one of these environment variables are set, the driver looks for `locales.dat` in the `$SYBASE/locales/locales.dat` directory to pick up the corresponding Adaptive Server character set. If the file is not found, the driver looks into its own map in memory to lookup the corresponding Adaptive Server character set.

# Adaptive Server ODBC Driver Configuration

When connecting to the database, ODBC applications typically use ODBC datasources. An ODBC datasource is a set of connection parameters, stored in the registry or in a file. ODBC datasources on non-Windows platforms typically reside in an `ini` file. Most ODBC Driver Managers provide a GUI tool to configure ODBC Driver and datasources.

## Microsoft Windows

When you use the Sybase SDK installation program to install the Adaptive Server ODBC Driver, it registers the driver on the local machine. You can manually register the Adaptive Server ODBC Driver on Microsoft Windows using the **regsvr32** utility.

### Registering the Adaptive Server ODBC Driver

Use the **regsvr32** to register the Adaptive Server ODBC Driver on Microsoft Windows platform manually.

**Note:** You do not need to manually register the Adaptive Server ODBC Driver if you have used the Sybase SDK installation program to install Adaptive Server ODBC Driver.

1. To register Adaptive Server 32-bit on Microsoft Windows x86 32-bit:
   a) Change to the `%SYBASE%\DataAccess\ODBC\dll` directory, which contains the Adaptive Server ODBC Driver DLL.
   b) Run the **regsvr32** utility to create registry entries in the `HKEY_LOCAL_MACHINE \SOFTWARE\ODBC\ODBCINST.INI` key:

```
regsvr32 sybdrvodb.dll
```

2. To register Adaptive Server 64-bit on Microsoft Windows x86-64 64-bit:
   a) Change to the `%SYBASE%\DataAccess64\ODBC\dll` directory, which contains the Adaptive Server ODBC Driver DLL.
   b) Run the **regsvr32** utility to create registry entries in the `HKEY_LOCAL_MACHINE \SOFTWARE\ODBC\ODBCINST.INI` key:

   ```
   regsvr32 sybdrvodb64.dll
   ```

3. To register Adaptive Server ODBC Driver 32-bit on Microsoft Windows x86-64 64-bit:
   a) Change to the `%SYBASE%\DataAccess\ODBC\dll` directory, which contains the Adaptive Server ODBC Driver DLL.
   b) Run the **regsvr32** utility to create registry entries in the `HKEY_LOCAL_MACHINE \SOFTWARE\Wow6432Node\ODBC\ ODBCINST.INI key`:

   ```
   regsvr32 sybdrvodb.dll
   ```

   **Note:** To configure a datasource using Adaptive Server ODBC Driver 32-bit on Microsoft Windows x86-64 64-bit, use the 32-bit ODBC Data Source Administrator **odbcad32.exe** located by default at `C:\WINDOWS\SysWOW64\`.

### Configuring a Datasource

Set up a datasource.

1. Launch the ODBC Administrator.

   See the online help for your specific Microsoft Windows operating system for detailed instructions.
2. Select the **User DSN** tab. Click **Add**.
3. Choose **Adaptive Server Enterprise** from the list of drivers.
4. Click **Finish**.
5. Select the **General** tab and enter values for these fields:
   - `Data Source Name` – a name for your datasource
   - `Description` – a description for your datasource
   - `Server Name` – an Adaptive Server Enterprise host name
   - `Server Port` – an Adaptive Server Enterprise port number
   - `Database Name` – a database name
   - `Logon ID` – a user name to log in to the Adaptive Server Enterprise database
6. Select **Use Cursors** to open cursors for every **select** statement.
7. Complete the **Connection** and **Advanced** tabs as needed.
8. Click **OK** to save the changes.

### See also

- *Connection Parameters* on page 35

## UNIX

The unixODBC Driver Manager supports configuring drivers and datasources from a GUI as well as the command line.

Refer to the ODBC Driver Manager's documentation for instructions on the GUI tool and command line syntax.

**Note:** The Adaptive Server ODBC Driver and datasources that use this driver cannot be configured using the GUI tools from the unixODBC Driver Manager. You must use the command line interface.

When configuring the driver and datasources using the unixODBC Driver Manager command line tool, you must supply a template file. Sample templates are described in the following section. You can also find these templates in:

- Adaptive Server ODBC Driver 32-bit: `$SYBASE/DataAccess/ODBC/samples`
- Adaptive Server ODBC Driver 64-bit: `$SYBASE/DataAccess64/ODBC/samples`

An example of a driver template file:

```
[Adaptive Server Enterprise]
Description=Sybase ODBC Driver
Driver=/install dir/driver library name
FileUsage=-1
```

where:

- *install dir* is the path to the Adaptive Server ODBC Driver installation.
- *driver library name* is the name of the driver library.

### Installing the Adaptive Server ODBC Driver

Install the Adaptive Server ODBC Driver using the unixODBC command line tool. Execute:

```
# odbcinst -i -d -f driver template file
```

where `driver template file` is the complete path to the Adaptive Server ODBC Driver template file.

**Note:** In most cases, this command needs to be executed as the root user because it modifies the `odbcinst.ini` file that is owned by root.

### Configuring a Datasource

Use the unixODBC command line tool to configure a datasource for Adaptive Server ODBC Driver.

This is a datasource template:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
```

Execute:

```
#  odbcinst -i -s -f dsn template file
```

where `dsn template file` is the complete path to the Adaptive Server ODBC datasource template file. This creates entries for the datasource in the `odbc.ini` file.

**Note:** The exact command you need to configure ODBC datasources depends on the ODBC Driver Manager you are using.

#### *Sybase iAnywhere ODBC Driver Manager*

The Sybase iAnywhere ODBC Driver Manager uses the information provided in the `odbc.ini` file to locate the driver and other connection information. The ODBCINI variable defines the location of the `odbc.ini` file.

#### *Configuring the ODBC Driver and Datasource*

Configure an `odbc.ini` file to use the ODBC driver and a datasource manually.

1. Create an `odbc.ini` file:

   ```
   [sampledsn]
   Description=Sybase ODBC Data Source
   UserID=sa
   Password=
   Driver=complete_path_to_libsybdrvodb.so
   Server=sampleserver
   Port=4100
   Database=pubs2
   UseCursor=1
   ```

2. Set the ODBCINI environment variable to the complete path to the `odbc.ini` file.

## ODBC ini Files

The ODBC Driver Manager stores driver and datasource information in `ini` files or the system registry.

See the ODBC Driver Manager documentation for the exact path for these `ini` files.

### Microsoft Windows

The `odbc.ini` and `odbcinst.ini` files are located in the `c:\winnt` directory. The Microsoft ODBC Driver Manager looks up these files or the registry at runtime when an application requests a connection to a datasource.

### UNIX

Information about the ODBC Driver installed on the system is saved in the `odbcinst.ini` file. This file is typically located at `/etc/odbcinst.ini`.

The information about datasources is saved in one of these files:

- User datasource information, available only to that user, is saved in the `$HOME/.odbc.ini` file, where `$HOME` is the user home directory.
- System datasource information, available to any user on the system, is usually saved in the `/etc/odbc.ini` file. If the same datasource is defined in both files, the user datasource takes precedence.

The ODBC Driver Manager looks up these files at runtime when an application requests a connection to a datasource.

Refer to your ODBC Driver Manager documentation for the exact path for these `ini` files. Some Driver Manager use alternate locations.

If your application is not using ODBC Driver Manager and uses the Adaptive Server ODBC Driver directly, the `ini` file is searched differently: The Adaptive Server ODBC Driver first looks for a file named `odbc.ini` in the current working directory; if the file is not found or the datasource not found in the file, it looks for `$SYBASE/odbc.ini`.

If your application uses the Sybase iAnywhere ODBC Driver Manager, set the ODBCINI environment variable to the complete path to the `odbc.ini` file. By default, `odbc.ini` is located under `$SYBASE`.

## Connections Using a Datasource

ODBC applications usually use datasources on the client computer for each database you want to connect to.

You can store sets of Adaptive Server Enterprise connection parameters as an ODBC datasource, in either the system registry or `ini` files. If you have a datasource, your connection string can simply name the datasource by using the DataSourceName (**DSN**) connection parameter:

```
DSN=my data source
```

## Connection Parameters

The connection parameters other than the **DSN** parameter that can be supplied to the Adaptive Server ODBC Driver.

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `AlternateServers` | A list of comma-separated host:port pairs such as server1:port1,server2:port2,...,serverN:portN; When establishing a connection, the Adaptive Server ODBC Driver first connects to the host and port specified by the Server and Port properties before going through the list of hosts and ports listed in AlternateServers. | No | Empty |
| `AnsiNull` | Strict ODBC compliance where you cannot use = NULL Instead, you must use IsNull. | No | 1 |
| `ApplicationName` | The name used by Adaptive Server to identify the client application. | No | Empty |
| `AuthenticationClient` | The type of client library to be used for Kerberos Authentication.<br>Valid values are:<br>• activedirectory<br>• cybersafekerberos<br>• mitkerberos | No | Empty |
| `AutoCommit` | Set the `autocommit` state.<br>Valid values are:<br>• 0 – `autocommit` is off (equivalent to setting **SQL_ATTR_AUTOCOMMIT** to SQL_AUTOCOMMIT_OFF)<br>• 1 – (default) `autocommit` is on (equivalent to setting **SQL_ATTR_AUTOCOMMIT** to SQL_AUTOCOMMIT_ON) | No | 1 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| AdjustLargePrecisionAndScale | Set the AdjustLargePrecisionAndScale state.<br>Valid values are:<br>• 0 – (default) AdjustLargePrecisionAndScale is on. Causes the Adaptive Server ODBC Driver to accept the calls made to SQLSetDescField() API to set precision or scale.<br>• 1 – AdjustLargePrecisionAndScale is off. The Adaptive Server ODBC Driver ignores any calls made to SQLSetDescField() API to set precision or scale, and uses the precision and scale of actual data value. | No | 0 |
| BackEndType | Specifies the target type of the datasource you are defining. The Adaptive Server ODBC Driver can communicate with multiple target objects, including database systems such as Adaptive Server, and gateways to non-Sybase database systems.<br>Valid values include:<br>• ASE<br>• DC DB2 Access Service<br>• DC TRS<br>• MFC Gatewayless<br>• Replication Server | No | ASE |
| BufferCacheSize | Keeps the input / output buffers in pool. When large results will occur, increase this value to boost performance. | No | 20 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| CancelQueryOnFreeStmt | Set the CancelQueryOnFreeStmt connection property.<br>Valid values are:<br>• 0 – (default) CancelQueryOnFreeStmt is off. No change in Adaptive Server ODBC Driver behavior.<br>• 1 – CancelQueryOnFreeStmt is on. When this is set to 1, SQLFreeStmt(handle, SQL_CLOSE) sends a cancel to ASE allowing large results to be closed more quickly | No | 0 |
| ChangeBigIntDefault | Specifies the default C type for bigint columns.<br>Valid values are:<br>• 0 – SQL_C_SBIGINT/SQL_CUBIGINT<br>• 1 – SQL_C_CHAR<br>• 2 – SQL_C_WCHAR | No | 0 |
| CharSet | Specifies the character set that is used to communicate to Adaptive Server. The valid values are ServerDefault, ClientDefault, NoConversions. | No | ServerDefault on Windows platforms, ClientDefault on all other platforms. |
| ClientCharset | Specifies the client character set. | No | The character set currently used by the operating system. |
| ClientHostName | The name of the client host passed in the login record to the server. | No | Empty |
| ClientHostProc | The identity of client process on this host machine passed in the login record to the server. | No | Empty |
| CodePageType | Specifies the type of character encoding used. The valid values are ANSI, OEM, and Other. | No | ANSI |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| CommandTimeOut | The time, in seconds, that a client has to wait for a command to execute. If a command does not execute within the time given, the client cancels the command and generates an error. | No | 0. A value of 0 indicates no time limit, allowing client to execute a command indefinitely its until return. |
| ConnectionTimeOut | The time, in seconds, that a client has to wait to establish a connection. If a connection is not established within the time given, the client cancels the attempt and generates an error. | No | 0. A value of 0 indicates no time limit, allowing ODBC to wait indefinitely for a database connection to be established. |
| CRC | By default, the driver returns the total records updated when multiple **update** statements are executed in a stored procedure. This count will also include all updates happening as part of the triggers set on an update or an insert.<br>Set this property to 0 if you want the driver to return only the last update count. | No | 1 |
| Database | The database to which you want to connect. | No | Empty |
| DataIntegrity | Enables Kerberos Data Integrity. | No | 0 (disabled) |
| DistributedTransactionProtocol | Sets the protocol to be used for distributed transactions. Valid values are XA (default) and OLE. | No | XA |
| DSPassword | The password used to authenticate on the LDAP server, if the LDAP server does not allow anonymous access. The password can be specified in the Directory Service URL (DSURL) as well. | No | Empty |
| DSPrincipal | The user name used to authenticate on the LDAP server, if the LDAP server does not allow anonymous access. The principal can be specified in the DSURL as well. | No | Empty |
| DSURL | The URL to the LDAP server. | No | Empty |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `DTCProtocol` (Microsoft Windows only) | Allows the driver to use either an XA protocol or OleNative protocol when using distributed transactions. | No | XA |
| `DynamicPrepare` | When set to 1, the driver sends **SQLPrepare** calls to Adaptive Server to compile/prepare. This can boost performance if you use the same query repeatedly. | No | 0 |
| `EnableBulkLoad` | Specifies whether bulk-load support is enabled:<br>• 0 – bulk-load support is disabled.<br>• 1 – bulk-load using **array insert** is enabled.<br>• 2 – bulk-load using the **bulk copy** interface is enabled.<br>• 3 – bulk-load using the fast logged **bulk copy** interface is enabled.<br>Use the Sybase-specific **SQL_ATTR_ENABLE_BULK_LOAD** connection attribute to set `EnableBulkLoad` programmatically. The attribute accepts the same values as EnableBulkLoad. | Yes | 0 |
| `EnableLOBLocator` | Specifies whether large object (LOB) locator support is enabled:<br>• 0 – LOB locator support is disabled.<br>• 1 – LOB locator support is enabled. | No | 0 |
| `EnableMDACheck` | Sets the checking mode for MDA scripts installed on the server.<br>Valid values are:<br>• 0 – disables MDA script checking.<br>• 1 – raise warning if the MDA script version is older than the driver version and continue with the connection.<br>• 2 – raise error if the MDA script version is older than the driver version and fail the connection. | No | 0 |
| `EnableServerPacketSize` | Allows Adaptive Server server versions 15.0 and later to choose the optimal packet size. | No | 1 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `Encryption` | The designated encryption.<br>Possible values: ssl. | No | Empty |
| `EncryptPassword` | Specifies whether password is transmitted in an encrypted format:<br>• 0 – use plain text password.<br>• 1 – use encrypted password. If it is not supported, return an error message.<br>• 2 – use encrypted password. If it is not supported, use plain text password.<br>When password encryption is enabled, and the server supports asymmetric encryption, this format is used instead of symmetric encryption. | No | 0 |
| `Escape` | Sets the ODBC escape character. | No | '\' |
| `FetchArraySize` | Specifies the number of rows the driver retrieves when fetching results from the server. | No | 25 |
| `HASession` | Specifies if high availability is enabled. 0 indicates high availability disabled, 1 high availability enabled. | No | 0 |
| `HomogeneousBatch` | Specifies the parameter batch handling mode:<br>• 0 – disables parameter batching.<br>• 1 – enables Adaptive Server parameter batching protocol.<br>• 2 – enables Adaptive Server bulk insert protocol.<br>Use the Sybase-specific **SQL_ATTR_HOMOGENEOUS_BATCH** connection attribute to set `HomogeneousBatch` programmatically. The attribute accepts the same values as `HomogeneousBatch`. | No | 0 |
| `IgnoreErrorsIfRSPending` | Specifies whether the driver is to continue processing or stop if error messages are present.<br>When set to 1, the driver ignores errors and continues processing the results if more results are available from the server. When set to 0, the driver stops processing the results if an error is encountered even if there are results pending | No | 0 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `InitializationString` | Sets a Transact-SQL statement to be executed at login. | No | Empty |
| `Isolation` | Specified the initial isolation level for the connection.<br>Valid values are:<br>• 0 – read uncommitted<br>• 1 – read committed<br>• 2 – repeatable read<br>• 3 – serializable | No | 0 |
| `Language` | The language in which Adaptive Server returns error messages. | No | Empty – Adaptive Server uses English by default |
| `LoginTimeOut` | Number of seconds to wait for a login attempt before returning to the application. If set to 0, the timeout is disabled, and a connection attempt waits for an indefinite period of time. | No | 15 |
| `NormalizeWCharParams` | Specifies whether to enable Unicode string normalization.<br>Set this property to 1 when the Adaptive Server configuration option `enable unicode normalization` is set to 0. Valid values are:<br>• 0 – disables Unicode string normalization.<br>• 1 – enables Unicode string normalization. | No | 0 |
| `OldPassword` | The current password. If `OldPassword` contains a value that is not null or an empty string, the current password is changed to the value contained in PWD. | No | Empty |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| PacketSize | The number of bytes per network packet transferred between Adaptive Server and the client. | No | Server-determined when driver is connected to Adaptive Server 15.0 or later. For older versions, the default is 512. |
| ParamsetsBeforeThread | Specifies the number of parameter sets to send before starting the response thread during a batch operation. | No | 50 |
| Port | The port number of Adaptive Server. | Yes | Empty |
| ProgName | Sets the value of progname to be used during login.<br>The specified value is truncated to 30 characters. | No | Empty |
| ProtocolCapture | Enable this property to capture TDS packets exchanged between an ODBC application and the server for debugging purposes. This property is enabled by specifying the capture file prefix. | No | Empty |
| PWD, Password | Contains the value of the password.<br>When performing a normal login, OldPassword is not set and PWD contains the value of the current password. When changing the password, OldPassword is set to the current password, and PWD contains the value of the new password. | No, if the user name does not require a password | Empty |
| QuotedIdentifier | Specifies if Adaptive Server treats character strings enclosed in double quotes as identifiers:<br>• 0 – does not enable quoted identifiers.<br>• 1 – enables quoted identifiers. | No | 0 |
| ReadWriteUnknown | When set, the columns are not updatable are marked as read/write unknown. | No | 0 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `ReleaseLocksOnCursorClose` | Specifies if Adaptive Server releases shared read-only cursor locks at isolation levels 2 and 3 when a cursor is closed:<br>• 0 – does not enable shared cursor locks release on close.<br>• 1 – enables shared cursor locks release on close. | No | 0 |
| `RemotePwd` | Sets the remote password(s) for servers in `servername,password;servername,password;...` format. | No | Empty |
| `ReplayDetection` | Enables Kerberos Replay Detection. | No | 0 |
| `RestrictMaximum PacketSize` | If there are memory constraints when `EnableServerPacketSize` is set to 1, set this property to an `int` value in multiples of 512 to a maximum of 65536. | No | 0 |
| `RetryCount`, `RetryDelay` | Control the connection retry behavior.<br>RetryCount is the number of times to attempt to connect to the server before reporting the connection failed. Between each retry, the driver delays for RetryDelay number of seconds.<br>By default, the ODBC application does not retry the connection. | No | 0 |
| `SecondaryPort` | The port number of the Adaptive Server acting as a failover server in an active-active or active-passive setup. | Yes, if HASession is set to 1. | Empty |
| `SecondaryServer` | The name or the IP address of the Adaptive Server acting as a failover server in an active-active or active-passive setup. | Yes, if HASession is set to 1. | Empty |
| `Server` | The name or IP address of the Adaptive Server. | Yes | Empty |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| ServerInitiated Transactions | When **SQL_ATTR_AUTOCOMMIT** is set to 1, Adaptive Server starts managing transactions as needed. The driver issues a **set chained on** command on the connection. Older ODBC drivers do not use this feature and manage the job of starting transactions by calling **begin tran**. Set this property to 0 to maintain the old behavior or require that your connection not use "chained" transaction mode. | No | 1 |
| ServerPrincipal | The logical name or the principal Adaptive Server name as configured in the key distribution center (KDC). Adaptive Server ODBC Driver uses the information to negotiate Kerberos authentication with the configured KDC and Adaptive Server. | No | Empty |
| ServiceName | Specifies the service name used to connect to the host. ServiceName can hold any string value. | No | Empty |
| SuppressParamFormat | Specifies that Adaptive Server is to suppress parameter format metadata when prepared statements are re-executed in a session.<br>Valid values are:<br>• 0 – the parameter format metadata is not suppressed.<br>• 1– the default value; Adaptive Server will not send parameter format metadata where possible. | No | 1 |
| SuppressRowFormat | Specifies that Adaptive Server is to suppress row format metadata (TDS_ROWFMT or TDS_ROWFMT2) for queries that are re-executed in a session.<br>Valid values are:<br>• 0 – the row format metadata is not suppressed.<br>• 1– the default value; Adaptive Server will not send row format metadata where possible. | No | 1 |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| SuppressRowFormat2 | Specifies that Adaptive Server is to send data using the TDS_ROWFMT byte sequence where possible instead of the TDS_ROWFMT2 byte sequence.<br>Valid values are:<br>• 0 – TDS_ROWFMT2 is not suppressed.<br>• 1– the default value; forces the server to send data in TDS_ROWFMT where possible. | No | 0 |
| SupressTDSControl Tokens | When set, the server does not send TDS control tokens. | No | 0 |
| SkipRowCountResults | The SkipRowCountResults connection property can be used to control how the ODBC Driver treats statements that return row count results.<br>• 0 – Adaptive Server ODBC Driver stops at each result set or row count. After executing a batch of statements using **SQLExecDirect** or **SQLExecute**, the application is positioned on the first available result which can be either a result set or a row count.<br>• 1– the default value; Adaptive Server ODBC Driver skips any row count results. After executing a batch of statements using **SQLExecDirect** or **SQLExecute**, the ODBC application is positioned on the first result set. | No | 1 |
| TextSize | The maximum size of binary or text data that can be sent over the wire. | No | Empty – Adaptive Server default is 32K. |
| TightlyCoupled Transaction (Microsoft Windows only) | When using distributed transactions, if you are using two DSNs that connect to the same Adaptive Server, set this to 1. | No | 0 |
| TrustedFile | If encryption is set to ssl, set this property to the path to the Trusted File. | No | Empty |
| UID, UserID | A case-sensitive user ID required to connect to the Adaptive Server. | Yes | Empty |

| Property Names | Description | Required | Default Value |
|---|---|---|---|
| `UseCursor` | Specifies which cursor type is to be used for SQL statements that generate result sets.<br>• 0 – use client-side cursors for all cases.<br>• 1 – use server-side cursors for all cases.<br>• 2 – use server-side cursors only when **SQLSetCursorName** ODBC function is called. | No | 0 |
| `WindowsCharsetConverter` | Allows the users to select which conversion library to use.<br>• 0 – the Adaptive Server ADO.NET Data Provider will utilize Unilib library for character set conversions.<br>• 1 – the Adaptive Server ADO.NET Data Provider will use the Microsoft Unicode conversion routines for character-set data conversion on Windows operating systems. | No | 0 |

**See also**

- *Adaptive Server Cluster Edition Features Supported* on page 50
- *Support for Mainframe Connect and DirectConnect for z/OS Option* on page 61
- *Character Sets* on page 29
- *Distributed Transactions* on page 53
- *Adaptive Server Features Supported* on page 49
- *Bulk-Load Support* on page 58
- *Large Object (LOB) Locator Support* on page 88
- *TDS Protocol Capture* on page 78
- *UseCursor Connection Property* on page 14
- *Suppressing Parameter Format Metadata* on page 86
- *Suppressing Row Format Metadata* on page 85
- *Suppressing Additional Row Format Information* on page 85

## ODBC Driver Version Information Utility

The **odbcversion** utility displays information about the ODBC driver.

*Syntax*

**odbcversion -version** | **-fullversion** | **-connect** *dsn userid password*

### *Parameters*

**-version** – displays a simple numeric version string for the ODBC driver.

**-fullversion** – displays the verbose version string for the ODBC driver.

**-connect** *dsn userid password* – displays the Adaptive Server version and the version of ODBC and OLEDB MDA scripts installed on that Adaptive Server.

Three variables are required with this parameter: *dsn*, which is the data source name for the Adaptive Server, and the user ID and password used to connect to the Adaptive Server.

### *Example*

Obtains the simple numeric version string of an ODBC driver used to connect to Adaptive Server:

```
odbcversion -version
```

The utility returns a numeric version string:

```
15.05.00.1015
```

### *Usage*

When no parameters are specified, the **odbcversion** utility displays a list of valid parameters.

# Adaptive Server Features Supported

Review the advanced Adaptive Server features that you can use with the Adaptive Server ODBC Driver.

## Microsecond Granularity for Time Data

Adaptive Server ODBC Driver provides microsecond-level precision for time data by supporting the SQL datatypes `bigdatetime` and `bigtime`.

`bigdatetime` and `bigtime` function similarly to and have the same data mappings as the SQL `datetime` and `time` datatypes:

- `bigdatetime` – corresponds to the Adaptive Server `bigdatetime` datatype and indicates the number of microseconds that have passed since January 1, 0000 00:00:00.000000. The range of legal `bigdatetime` values is from January 1, 0001 00:00:00.000000 to December 31, 9999 23:59:59.999999.
- `bigtime` – corresponds to the Adaptive Server `bigtime` datatype and indicates the number of microseconds that have passed since the beginning of the day. The range of legal `bigtime` values is from 00:00:00.000000 to 23:59:59.999999.

*Usage*

- When connecting to Adaptive Server 15.5, the Adaptive Server ODBC Driver transfers data using the `bigdatetime` and `bigtime` datatypes, even if the receiving Adaptive Server columns are defined as `datetime` and `time`.

  This means that Adaptive Server may silently truncate the values from the Adaptive Server ODBC Driver to fit the Adaptive Server columns. For example, a `bigtime` value of 23:59:59.999999 is saved as 23:59:59.996 in an Adaptive Server column with datatype `time`.
- When connecting to Adaptive Server 15.0.x and earlier, the Adaptive Server ODBC Driver transfers data using the `datetime` and `time` datatypes.

## Asynchronous Execution for ODBC

By default, drivers execute ODBC functions synchronously. That is, the application calls a function and the driver returns control to the application when execution is complete.

With asynchronous execution, the driver returns control to the application after minimal processing and before execution is complete. This allows the application to execute in parallel other functions while the first function is still executing. Asynchronous execution is beneficial when a task is complex and requires a significant amount of time to execute.

The Adaptive Server ODBC Driver by Sybase supports a maximum of one concurrent statement in asynchronous mode. Only one concurrent statement, synchronous or asynchronous, can be executed if server-side cursors are used or if the connection's auto-commit is disabled.

To use connection-level asynchronous execution with the Adaptive Server ODBC Driver by Sybase, call **SQLSetConnectAttr** and set **SQL_ATTR_ASYNC_ENABLE** to SQL_ASYNC_ENABLE_ON.

For more information about asynchronous execution and its application, refer to the *ODBC Programmer's Reference* that is available at the *Microsoft Developers Network* at *http://msdn.microsoft.com/*.

**Note:** Calling **SQLCancel** when no processing is being done will not close the associated cursors. ODBC applications should explicitly call **SQLFreeStmt** or **SQLCloseCursor** to close cursors.

# Adaptive Server Cluster Edition Features Supported

Learn about the Adaptive Server ODBC Driver features that support the Cluster Edition, where multiple Adaptive Servers connect to a shared set of disks and a high-speed private interconnection.

This allows Adaptive Server to scale using multiple physical and logical hosts.

See the *Adaptive Server Enterprise Clusters Users Guide*.

## Login Redirection

The login redirection occurs during the login sequence and the client application does not receive notification that it was redirected.

At any given time, some servers within a Cluster Edition environment are usually more loaded with work than others. When a client application attempts to connect to a busy server, the login redirection feature helps balance the load of the servers by allowing the server to redirect the client connection to less busy servers within the cluster. Login redirection is enabled automatically when a client application connects to a server that supports this feature.

**Note:** When a client application connects to a server that is configured to redirect clients, the login time may increase because the login process is restarted whenever a client connection is redirected to another server.

## Connection Migration

The connection migration feature allows a server in a Cluster Edition environment to dynamically distribute load, and seamlessly migrate an existing client connection and its context to another server within the cluster.

This feature enables the Cluster Edition environment to achieve optimal resource utilization and decrease computing time. Because migration between servers is seamless, the connection migration feature also helps create a high availability (HA), zero-downtime environment. Connection migration is enabled automatically when a client application connects to a server that supports this feature.

**Note:** Command execution time may increase during server migration. Sybase recommends that you increase the command timeouts accordingly.

## Connection Failover in Cluster Edition

Connection failover allows a client application to switch to an alternate Adaptive Server if the primary server becomes unavailable due to an unplanned event, like power outage or a socket failure.

In the Adaptive Server Cluster Edition, client applications can failover numerous times to multiple servers using dynamic failover addresses.

With high availability enabled, the client application does not need to be configured to know the possible failover targets. Adaptive Server keeps the client updated with the best failover list based on cluster membership, logical cluster usage and load distribution. During failover, the client refers to the ordered failover list while attempting to reconnect. If the driver successfully connects to a server, the driver internally updates the list of host values based on the list returned. Otherwise, the driver throws a connection failure exception.

### See also

- *Failover in High Availability Systems* on page 69

### Enabling Connection Failover Using the Adaptive Server ODBC Driver User Interface (Windows)

Enable the Cluster Edition connection failover in the Adaptive Server ODBC Driver through its user interface.

1. Open the Adaptive Server Enterprise dialog box.
2. Go to the **Connection** tab.
3. Select **Enable High Availability**.
4. (optional) Enter `alternate servers` and `ports` in the **Alternate Servers** field using this format:

```
server1:port1,server2:port2,...,serverN:portN;
```

In establishing a connection, the Adaptive Server ODBC Driver first attempts to connect to the primary host and port defined in the General tab of the Adaptive Server Enterprise dialog box. If Adaptive Server ODBC Driver fails to establish a connection, it then searches through the list of hosts and ports specified in the **Alternate Servers** field.

### Enabling Connection Failover Using the Adaptive Server ODBC Driver Connection String

Use connection string to enable the connection failover in the Adaptive Server ODBC Driver.

Set the `HASession` connection string property to 1.

You can use `SQLDriverConnect` to specify a connection string. For example:

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;HASession=1;
AlternateServers=server2:port2,...,serverN:portN;
```

The preceding example defines server1 and port1 as the primary server and port. If Adaptive Server ODBC Driver fails to establish connection to the primary server, and alternate servers are defined, it searches through the ordered list of servers and ports specified in the **Alternate Servers** field until a connection is established or until the end of the list is reached.

### Enabling Connection Failover Using the unixODBC Driver Manager (UNIX)

Enable the connection failover using the unixODBC command line tool.

If you are linking to the unixODBC Driver Manager, edit the Adaptive Server ODBC datasource template, `odbc.ini`, and reinstall the datasource using the unixODBC command line tool:

```
# odbcinst -i -s -f dsn_template_file
```

where *dsn_template_file* is the complete path to the Adaptive Server ODBC datasource template file.

### Enabling Connection Failover Using the Adaptive Server ODBC Driver or the Sybase iAnywher ODBC Driver Manager

Use the Adaptive Server Driver Manager or the Sybase iAnywhere ODBC Driver Manager to enable connection failover.

If you are directly linking to the Adaptive Server ODBC Driver or the Sybase iAnywhere ODBC Driver Manager, modify the `odbc.ini` file to add the alternate servers.

For example:

```
ODBC Data Source UserID=sa
Password= Driver=Adaptive
Server Enterprise Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
```

```
HASession=1
AlternateServers=server2:port2,server3:port3;
```

**Note:** The list of alternate servers specified in the GUI or the connection string is used only during initial connection. After the connection is established with any available instance, and if the client supports high availability, the client receives an updated list of the best possible failover targets from the server. This new list overrides the specified list.

# Distributed Transactions

Use Adaptive Server ODBC Driver to participate in two-phase commit transactions.

This feature is supported only on Microsoft Windows and requires that Microsoft Distributed Transaction Coordinator (MS DTC) be the transaction coordinator managing two-phase commit.

Sybase supports all of these programming models:

- Applications using MS DTC directly
- Applications using Sybase EAServer
- Applications using Microsoft Transaction Server (MTS) or COM+

## Programming for Microsoft Distributed Transaction Coordinator (MS DTC)

Use the **DtcGetTransactionManager** function to program an ODBC application.

1. Connect to MS DTC by using the **DtcGetTransactionManager** function.

   For information about MS DTC, see *Microsoft Distributed Transaction Coordinator* documentation.
2. Call **SQLDriverConnect** or **SQLConnect** once for each Adaptive Server connection to establish.
3. Call the **ITransactionDispenser::BeginTransaction** function to begin an MS DTC transaction and to obtain an OLE Transaction object that represents the transaction.
4. Call **SQLSetConnectAttr** one or more times for each ODBC connection you want to enlist in the MS DTC transaction.

   **SQLSetConnectAttr** must be called with an attribute of `SQL_ATTR_ENLIST_IN_DTC` and a `ValuePtr` of the Transaction object (obtained in step 3).
5. Call **SQLExecDirect** one or more times for each **insert** or **update** SQL statement.
6. Call the **ITransaction::Commit** function to commit the MS DTC transaction. The Transaction object is no longer valid.

To perform a series of MS DTC transactions, repeat steps 3 through 6.To release the reference to the Transaction object, call the **ITransaction::Release** function.To use an ODBC connection with an MS DTC transaction and then use the same connection with a local

Adaptive Server transaction, call **SQLSetConnectAttr** with a `ValuePtr` of `SQL_DTC_DONE` to unenlist the connection from the transaction.

**Note:** Also, you can call **SQLSetConnectAttr** and **SQLExecDirect** separately for each Adaptive Server, instead of calling them as suggested in steps 4 and 5.

## Programming Components Deployed in Sybase EAServer, MTS or COM+

Create components that participate in distributed transactions in Sybase EAServer, MTS, or COM+.

1. Call **SQLDriverConnect** once for each Adaptive Server connection you want to establish.
2. Call **SQLExecDirect** once for each **insert** or **update** SQL statement.
3. Deploy your component to MTS, and configure the transaction attributes as needed.

The transaction coordinator creates a distributed transaction as needed, and the component that uses the Adaptive Server ODBC Driver automatically enlists in the global transaction. Then, the transaction coordinator commits or rolls back the distributed transaction.

## Connection Properties for Distributed Transaction Support

The connection properties for a distributed transaction.

- Distributed Transaction Protocol (`DistributedTransactionProtocol`) – to specify the protocol used to support the distributed transaction, either XA Interface standard or MS DTC OLE Native protocol, select the `Distributed Transaction Protocol` in the ODBC Data Source dialog, or set the property `DistributedTransactionProtocol` = *OLE* native protocol in the connection string. The default is *XA*.
- Tightly Coupled Transaction (`TightlyCoupledTransaction`) – when a distributed transaction using two resource managers points to the same Adaptive Server, it is a Tightly Coupled Transaction. Under these conditions, if you do not set this property to 1, the distributed transaction may fail.

To summarize, if you open two database connections to the same Adaptive Server and then enlist these connections in the same distributed transaction, you must set `TightlyCoupledTransaction=1`. To set this property, select the Tightly Coupled Transaction in the ODBC Data Source dialog box, or pass the property `TightlyCoupledTransaction=1` in the connection string.

**Warning!** Enlistment with **SQLSetConnectAttr** returns a `SQL_ERROR` if the connection has already begun a local transaction, either by using **SQLSetConnectAttr** with the `SQL_AUTOCOMMIT_OFF` or by executing the **BEGIN TRANSACTION** statement explicitly using **SQLExecDirect**.

# Directory Services

Directory services allow the Adaptive Server ODBC Driver to get connection and other information from a central LDAP server; then, it uses this information to connect to an Adaptive Server. It uses a property called Directory Service URL (`DSURL`), that indicates which LDAP server to use.

## LDAP as a Directory Service

Lightweight Directory Access Protocol (LDAP) is an industry standard for accessing directory services.

Directory services allow components to look up information by a distinguished name (DN) from an LDAP server that stores and manages server, user, and software information that is used throughout the enterprise or over a network.

LDAP defines the communication protocol and the contents of messages exchanged between clients and servers. The LDAP server can store and retrieve information about:

- Adaptive Server, such as IP address, port number, and network protocol
- Security mechanisms and filters
- High-availability companion server name

See *Adaptive Server Enterprise System Administration Guide*.

You can use these access restrictions when configuring the LDAP server:

- Anonymous authentication – all data is visible to any user.
- User name and password authentication – Adaptive Server uses the default user name and password from the file.

User name and password authentication properties establish and end a session connection to an LDAP server.

**Note:** The LDAP server can be located on a different platform from the one on which Adaptive Server or the clients are running.

## Using Directory Services

Use **ConnectString** to define directory services properties.

To use directory services, add these properties to **ConnectString**:

```
DSURL=ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

The URL is an LDAP URL and uses LDAP libraries to resolve the URL.

To support high availability on the LDAP server, the DSURL accepts multiple URLs, separated by a semicolon:

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?
sybaseServername=MANGO};
```

The provider attempts to get the properties from the LDAP servers in the order specified. For example:

```
ldap://hostport/dn[?attrs[?scope[?filter[?userdn?userpass]]]]
```

where:

- *hostport* is a host name with an optional `portnumber`, for example, `SYBLDAP1:389`.
- *dn* is the search base, for example, `dc=sybase,dc-com`.
- *attrs* is a comma-separated list of attributes requested from the LDAP server. You must leave it blank. Data Provider requires all attributes.
- *scope* is one of three strings:
  - *base* (the default) searches the base.
  - *one* searches immediate children.
  - *sub* searches the sub-tree.
- *filter* is the search filter, which is, generally, the **sybaseServername**. You can leave the search filter blank and set the `datasource` or `Server Name` property in the **ConnectionString**.
- *userdn* is the user's distinguished name (dn). If the LDAP server does not support anonymous login, you can set the user's dn here, or you can set the `DSPrincipal` property in the **ConnectionString**.
- *userpass* is the password. If the LDAP server does not support anonymous login, you can set the password here, or you can set the `DSPassword` property in the **ConnectionString**.

The URL can contain `sybaseServername`, or you can set the property `Server Name` to the service name of the LDAP Sybase server object.

These properties are useful when using Directory Services:

- `DSURL` – set to LDAP URL. The default is an empty string.
- `Server` – the service name of the LDAP Sybase server object. The default is an empty string.
- `DSPrincipal` – the user name to log in to the LDAP server if it is not a part of `DSURL` and the LDAP server does not allow anonymous access.
- `DSPassword` or `Directory Service Password` – the password to authenticate on the LDAP server if it is not a part of `DSURL` and the LDAP server does not allow anonymous access.

## Enabling Directory Services

Enable directory services on the platform you are using.

### Enabling Directory Services on Microsoft Windows

Use the ODBC DataSource Administrator to enable directory services on Microsoft Windows platform.

1. Launch the ODBC DataSource Administrator.
2. Select the datasource that you want to use and choose **Configure**.
3. Click the **Connection** tab.
4. In the **Directory Service Information** group, provide the complete `URL` in the **URL** field. You can also provide the `user name` in the **User ID** field and the `LDAP Service Name` in the **Service Name** field, to log in to the LDAP server.

This is an example of the attributes you must specify in the connection string:

```
DSURL = ldaps://huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secret
DSServiceName = myAse
```

The Certificate Authority signing certificate used to sign the LDAP server's certificate must be installed in the Microsoft Certificate Store.

### Enabling Directory Services for Linux

Use the **openladap** packages to enable directory services on Linux platform.

#### Prerequisites

Install these packages:

- **openldap**-2.0 (runtime)
- **openldap-devel**-2.0

The Adaptive Server ODBC Driver attempts to load a file named `libldap.so`, but to create a symbolic link with this file, you must install the **openldap-devel** package. The **openldap** runtime package does not create the symbolic link.

If you are linking to the unixODBC Driver Manager:

#### Task

1. Edit the Adaptive Server ODBC datasource template, `odbc.ini`.
2. Reinstall the datasource using the **unixODBC** command line tool:

```
# odbcinst -i -s -f <dsn template file>
```

where *dsn template file* is the complete path to the Adaptive Server ODBC datasource template file.

If you are directly linking to the Adaptive Server ODBC Driver, modify the `odbc.ini` file. For example:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password= Driver=Adaptive
Server Enterprise Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
DSURL=ldap://SYBLDAP1:389/dc=sybase,dc=com??one?
sybaseServername=MANGO
```

When LDAPS is specified in the LDAP URL instead of LDAP, an SSL connection is established. The application needs to set the TrustedFile attribute to indicate where the Certificate Authority signing certificate is located.

Example of the attributes that you must specify for the DSN in `odbc.ini` (or connection string):

```
DSURL = ldaps:// huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secretDSServiceName = myAse
```

```
TrustedFile = /usr/u/sybase/config/trusted.txt
```

**Note:** The Certificate Authority signing certificate used to sign the LDAP server's certificate must be appended to the `trusted.txt` file.

## Bookmark and Bulk Support

Sybase supports bookmarks and SQL bulk operations for the ODBC Driver.

Bulk insertions that use **SQLBulkOperations** with the option of **SQL_ADD** and cursor positioned updates and deletions using **SQLSetPos** (**SQL_UPDATE**, **SQL_DELETE**, **SQL_POSITION**). For instructions on using **SQL_ADD** and **SQLSetPos**, refer to the *ODBC Programmer's Reference* found in the *Microsoft Developer Network* library at *http://msdn.microsoft.com*.

## Bulk-Load Support

The Adaptive Server ODBC Driver supports bulk-load interface for fast insertions of large sets of rows to Adaptive Server.

This interface is invoked when **SQLBulkOperations** is used with the `SQL_ADD` option and the `EnableBulkLoad` connection property is set. Two types of bulk loading are supported:

- Array Inserts – you can use this type of bulk-loading within a single or multistatement transaction; the database connection can be set to **autocommit off**.
- Bulk Copy – this is supported only in single statement transactions, and you must to ensure that:
  - The database connection is set to `autocommit on`.
  - The `select into` or `bulkcopy` option on Adaptive Server is turned on.

  If the target table meets the criteria for high-speed version of **bulk copy**, Adaptive Server inserts the rows using this version of **bulk copy**.

  **Note:** Using the bulk copy mode with the `select into` or `bulkcopy` option enabled affects the recoverability of the database. After the **bulk copy** operation is complete, the system administrator must dump the database to ensure its future recoverability.

This table guides you on what bulk-load option to use.

**Table 1. Bulk-Load Option Usage**

| Use Case | Additional Consideration | Bulk-Load Option to Use | Note |
|---|---|---|---|
| Insertion of single or small number of rows. | | None | |
| Insertion of large batch of rows. | The batch is part of a multistatement transaction. | Array Inserts | Rows are inserted faster than when bulk load is disabled. |
| | You cannot enable the Adaptive Server `select into` or `bulkcopy` option because of recoverability considerations. | Array Inserts | Rows are inserted faster than when bulk load is disabled. |
| | The batch is a single transaction and the Adaptive Server `select into`or `bulkcopy` option is enabled. | Bulk Copy | Adaptive Server can use high-speed bulk copy, which is faster than array inserts. The performance of Bulk Copy is still slightly faster than Array Inserts even if high-speed bulk copy is not used. |

See the *Adaptive Server Enterprise Utility Guide* for information about the implications of enabling `select into` or `bulkcopy` and the conditions under which high-speed or logged **bulk copy** is used.

### *EnableBulkLoad Connection Property*

Enable or disable bulk-load support using the `EnableBulkLoad` connection property:

- 0 – the default value, which disables bulk load.
- 1 – enables bulk load using **array insert**.
- 2 – enables bulk load using the **bulk copy** interface.
- 3 – enables bulk load using the fast logged **bulk copy** interface.

Alternatively, use the Sybase-specific **SQL_ATTR_ENABLE_BULK_LOAD** connection attribute to set `EnableBulkLoad` programmatically. The attribute accepts the same values as `EnableBulkLoad`. For example:

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_ENABLE_BULK_LOAD,
(SQLPOINTER)3, SQL_IS_INTEGER);
```

*Performance Considerations*

Although this feature does not require special configuration on the server, a larger page size and network packet size significantly improves performance. Depending on the client memory, using larger batches also improves performance.

*Limitations*

Triggers are ignored on tables selected for bulk loading.

## Enabling Bulk Load Using the ODBC Data Source Administrator User Interface

Enable bulk load using the ODBC Datasource Administrator GUI.

1. Open the Data Source Name (DSN) Configure window from the ODBC Data Source Administrator.
2. Select the **Advanced** tab.
3. Select the appropriate option under **Enable Bulk Load**.

   The default value of `EnableBulkLoad` connection property is 0, which means **insert** commands are used.

## Enabling Bulk Load Using the ODBC Connection String

Use the **SQLDriverConnect** to enable the bulk-load support.

1. Use **SQLDriverConnect** to specify a connection string.
2. Set the `EnableBulkLoad` connection string property to 0, 1, 2, or 3, as appropriate.

   For example:

   ```
   Driver=AdaptiveServerEnterprise;server=server1;
   port=port1;UID=sa;PWD=;EnableBulkLoad=1;
   ```

# Support for data-at-exec in Adaptive Server ODBC Driver

In Adaptive Server ODBC Driver version 15.7 ESD #7, the data-at-exec feature has been enhanced to support bulk and batch operations resulting in lower memory utilization.

In earlier versions, all of the data for bound parameters had to be fully loaded before calling **SQLBulkOperations** or executing a batch. In ESD #7, the application does not need to preload any parameter data, it can be sent in chunks using **SQLPutData**. When using the Adaptive Server ODBC Driver batch protocol (**SQLExecute**/**SQLExecDirect** with **SQL_ATTR_BATCH_PARAMS**), data-at-exec is supported as long as **SQL_ATTR_PARAMSET_SIZE**is 1. Using data-at-exec for LOB columns requires the server to support LOB parameters.

# Support for Mainframe Connect and DirectConnect for z/OS Option

Adaptive Server ODBC Driver by Sybase supports Mainframe Connect DirectConnect™ for z/OS Option through the `ServiceName` and `BackEndType` configuration properties.

## ServiceName Configuration Property

The `ServiceName` property specifies the service name used to connect to the host. `ServiceName` can hold any string value. Its default value is an empty string ("").

## BackEndType Configuration Property

The `BackEndType` configuration property specifies the target type of the DSN you are defining.

The ODBC Driver can communicate with multiple targets including database systems like Adaptive Server and gateways to non-Sybase database systems.

The Adaptive Server ODBC Driver supports these backend types:

- ASE (default)
- MFC Gatewayless
- DC DB2 Access Service
- DC TRS
- Replication Server

### Replication Server Connection Support

Adaptive Server Enterprise ODBC Driver can connect to Replication Server to monitor and administer the server.

Only valid Replication Server Administration commands sent by the ODBC Driver are supported by Replication Server. Set the `BackEndType` connection property to `Replication Server` for Replication Server connections.

# DSN Migration Tool

The ODBC DSN Migration tool can help you migrate from the Data Direct ODBC driver to the Adaptive Server ODBC Driver by Sybase.

## Using the Migration Tool

The **dsnmigrate** tool uses switches to control DSN which are migrated.

From the command line, enter:

```
dsnmigrate.exe [/?|/help] [l|/ul|/sl][/a|/ua|/sa]
   [[/dsn|/udsn|/sdsn]=dsn] [/suffix=suffix]
```

All DSNs that are converted are renamed to "<dsn>-backup" before the conversion is completed.

When the new Sybase DSNs are created and the conversion is completed, the name is changed to"<dsn>," which allows existing applications to continue to run without any modifications.

## Conversion Switches

Switches used in the conversion.

**Table 2. Switches**

| Switches | Description of Results |
|---|---|
| /?,/h,/help | Lists and describes the switches. The list also appears if you call **dsnmigrate** with no command line arguments. |
| /l | Displays a list of all Sybase Data Direct user and system DSNs. |
| /ul | Displays a list of all Sybase Data Direct user DSNs. |
| /sl | Displays a list of all Sybase Data Direct system DSNs. |
| /a | Converts all Sybase Data Direct user and system DSNs. |
| /ua | Converts all Sybase Data Direct user DSNs. |
| /sa | Converts all Sybase Data Direct system DSNs. |

| Switches | Description of Results |
|---|---|
| /dsn | Converts specific Sybase Data Direct user or system DSNs. |
| /udsn | Converts specific Sybase Data Direct user DSNs. |
| /sdsn | Converts specific Sybase Data Direct system DSNs. |
| dsn | The name of the DSN to be converted. |
| /suffix | An optional switch that changes the way DSNs are named. If this switch is used, the original DSN is retained and the new DSN is named "<dsn>-<suffix>." |
| suffix | The suffix that is used to name the new DSN. |

# Password Encryption

By default, the Adaptive Server ODBC Driver sends plain-text passwords over the network to Adaptive Server for authentication.

However, the Adaptive Server ODBC Driver also supports symmetrical and asymmetrical password encryption; you can change the default behavior of and encrypt your password before it is sent over the network.

The symmetrical encryption mechanism uses the same key to encrypt and decrypt the password, whereas an asymmetrical encryption mechanism uses one key (the public key) to encrypt the password and another key (the private key) to decrypt the password. Because the private key is not shared across the network, the asymmetrical encryption is considered more secure than symmetrical encryption. When password encryption is enabled, and the server supports asymmetric encryption, this format is used instead of symmetric encryption.

You can encrypt login and remote passwords using the Sybase Common Security Infrastructure (CSI). CSI 2.6 complies with the Federal Information Processing Standard (FIPS) 140-2.

## Enabling Password Encryption

To enable password encryption, set the `EncryptPassword` connection property, which specifies whether the password is transmitted in encrypted format.

When password encryption is enabled, the password is sent over the wire only after a login is negotiated; the password is first encrypted and then sent.

The `EncryptPassword` values are:

- 0 – (default) use plain text password.
- 1 – use encrypted password. If encryption is not supported, return an error message.
- 2 – use encrypted password. If encryption is not supported, use plain text password.

**Note:** To use the password encryption feature, you must have a server that supports password encryption, such as Adaptive Server 15.0.2. Asymmetrical encryption requires additional processing time and may cause a slight delay in login time.

### Encrypting Passwords on Microsoft Windows

Use the `EncryptPassword` connection property for encrypting passwords on Microsoft Windows.

1. Launch the ODBC DataSource Administrator.
2. Select the datasource you want to use and choose **Configure**.
3. Click the **Advanced** tab.
4. Select **EncryptPassword**.

You can use the `EncryptPassword` connection property in a call to **SQLDriverConnect**.

**Note:** You can only use the user interface to set `EncryptPassword` to 0 or 1. To set `EncryptPassword` to 2, use a connection string.

### Encrypting Password on UNIX

Use the unixODBC Driver Manager or the Sybase iAnywhere Driver Manager to encrypt passwords on UNIX platform.

1. To link to the unixODBC Driver Manager, edit the datasource template and reinstall the datasource using the unixODBC command line tool:

   ```
   #  odbcinst -i -s -f dsn template file
   ```

   where *dsn template file* is the complete path to the Adaptive Server ODBC datasource template file.
2. If you are directly linking to the Adaptive Server ODBC Driver or the Sybase iAnywhere ODBC Driver Manager, modify the `odbc.ini` file.

   This is an example of an `odbc.ini` datasource template file:

   ```
   [sampledsn]
   Description=Sybase ODBC Data Source
   UserID=sa
   Password=
   ```

   ```
   Driver=Adaptive Server Enterprise
   Server=sampleserver
   Port=4100
   Database=pubs2
   UseCursor=1
   EncryptPassword=1
   ```

## Password Expiration Handling

Every company has a specific set of password policies for its database system. Depending on the policies, the password expires at a specific date and time.

Unless the password is reset, the Adaptive Server ODBC Driver connected to a database throws password expired errors and suggests that the user change the password using isql. The password expiration handling feature allows users to change their expired passwords using the Adaptive Server ODBC Driver.

### *Changing the Password Through the Connection String Properties*

Set these two connection string properties:

- `OldPassword` – the current password. If `OldPassword` contains a value that is not null or an empty string, the current password is changed to the value contained in `PWD`.
- `PWD` – contains the value of the password. If `OldPassword` is not null, `PWD` contains the value of the current password. If `OldPassword` does not exist, or is null, `PWD` contains the value of the new password.

### *Changing the Password Through a Dialog Box*

A change password dialog is activated when **SQLDriverConnect** with `SQL_DRIVER_PROMPT` is set to true. In this dialog, enter the current password and the new password.

## SSL Overview

Secure Sockets Layer (SSL) is an industry standard for sending wire- or socket-level encrypted data over client-to-server and server-to-server connections.

Before the SSL connection is established, the server and the client negotiate and agree upon a secure encrypted session. This is called the SSL handshake.

**Note:** Additional overhead is required to establish a secure session, because data increases in size when it is encrypted; it also requires additional computation to encrypt or decrypt information. Under normal circumstances, the additional I/O accrued during the SSL handshake can make user login 10 to 20 times slower.

### *SSL Handshake*

When a client application requests a connection, the SSL-enabled server presents its certificate to prove its identity before data is transmitted.

The SSL handshake consists of these steps:

1. The client sends a connection request to the server. The request includes the SSL (or Transport Layer Security, TLS) options that the client supports.
2. The server returns its certificate and a list of supported cipher suites, which includes SSL/TLS support options, the algorithms used for key exchange, and digital signatures. Cipher suites are preferential lists of key-exchange algorithms, hashing methods, and encryption methods used by the SSL protocol.
3. A secure, encrypted session is established when both client and server have agreed upon a cipher suite.

### *Cipher Suites*

During the SSL handshake, the client and server negotiate a common security protocol through a cipher suite.

By default, the strongest cipher suite supported by both the client and the server is the cipher suite used for the SSL-based session. Server connection attributes are specified in the connection string or through directory services such as LDAP.

The Adaptive Server ODBC Driver and Adaptive Server support the cipher suites that are available with the SSL Plus library API and the cryptographic engine, Security Builder, both from Certicom Corp.

---

**Note:** The following list of cipher suites conform to the Transport Layer Security (TLS) specification, which is an enhanced version of SSL 3.0, and an alias for the SSL version 3.0 cipher suites.

---

These are the cipher suites, ordered from the strongest to the weakest, supported in Adaptive Server OBDC Driver:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_RC4_128_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_DHE_DSS_WITH_DES_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

For more specific information about the SSL handshake and the SSL/TLS protocol, see the *Internet Engineering Task Force Web site* at *http://www.ietf.org*.

For a complete description of cipher suites, go to the *IETF organization Web site* at *http://www.ietf.org/rfc/rfc2246.txt*.

## SSL Security Levels in Adaptive Server ODBC Driver

Review SSL security levels in the Adaptive Server ODBC Driver.

In Adaptive Server ODBC Driver, SSL provides these levels of security:

- When the SSL session is established, user name and password are transmitted over a secure, encrypted connection.
- When establishing a connection to an SSL-enabled server, the server authenticates itself, proves that it is the server you intended to contact, and an encrypted SSL session begins before any data is transmitted.
- A comparison of the server certificate's digital signature can determine if any information received from the server was modified in transit.

## Validating the Server by its Certificate

Any Adaptive Server OBDC Driver client connection to an SSL-enabled server requires have a certificate file, which consists of the server's certificate and an encrypted private key.

The certificate must also be digitally signed by a signing/certification authority (CA). Adaptive Server OBDC Driver client applications establish a socket connection to Adaptive Server similar to the way that existing client connections are established. Before any user data is transmitted, an SSL handshake occurs on the socket when the network transport-level connect call completes on the client side and the accept call completes on the server side.

To make a successful connection to an SSL-enabled server:

1. The SSL-enabled server must present its certificate when the client application makes a connection request.
2. The client application must recognize the CA that signed the certificate. A list of all trusted CAs is in the "trusted roots file."

The trusted roots file

The list of known and trusted CAs is maintained in the trusted roots file. The trusted roots file is similar in format to a certificate file, except that it contains certificates for CAs known to the entity (such as client applications, servers, network resources, and so on). The system security officer adds and deletes trusted CAs using a standard ASCII-text editor.

The application program specifies the location of the trusted roots file using the `TrustedFile=trusted file path` property in the **ConnectString**. A trusted roots

file with the most widely used CAs (thawte, Entrust, Baltimore, VeriSign, and RSA) is installed in a file located at `$SYBASE/config/trusted.txt`.

For more information about certificates, see the *Open Client Client-Library/C Reference Manual*.

## Enabling SSL Connections

Use **ConnectionString** to enable SSL connection for Adaptive Server ODBC Driver.

To enable SSL for Adaptive Server ODBC Driver, add `Encryption=ssl` and `TrustedFile=<filename>` (where *filename* is the path to the `trusted roots` file) to the **ConnectString**. The Adaptive Server ODBC Driver then negotiates an SSL connection with the Adaptive Server.

**Note:** Adaptive Server must be configured to use SSL. For more information on SSL, see the *Adaptive Server Enterprise System Administration Guide*.

### Enabling SSL Connections on Microsoft Windows

Set the `TrustedFile` property in the connection string to the file name of the trusted roots file, before you enable SSL. The file name should contain the path to the file as well.

1. Set the `Encryption` property in the connection string to `ssl`.
2. Launch the ODBC DataSource Administrator.
3. Select the datasource name (DSN) you would like to use and choose **Configure**.
4. Click the **Connection** tab.
5. Select **UseSSL** in the Secure Socket Layer Group.
6. Provide the complete path to the trusted roots file in the **TrustedFile** field.

### Enabling SSL Connections on UNIX

Use the **odbcinst** utility to enable SSL connections.

1. Start the unixODBC Driver Manager **odbcinst** utility.
2. Open an existing datasource template or create a new one.
3. To the datasource template, add:

   ```
   Encryption=sslTrustedFile=<filename>line
   ```

4. Reinstall the datasource using:

   ```
   #  odbcinst -i -s -f dsn template file
   ```

   where *dsn template file* is the complete path to the Adaptive Server ODBC datasource template file.

If you are linking directly to the Adaptive Server ODBC Driver or the Sybase iAnywhere ODBC Driver Manager, modify the `odbc.ini` file.

This is an example of the `odbc.ini` datasource template file:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
Encryption=ssl
TrustedFile=<SYBASE>/config/trusted.txt
```

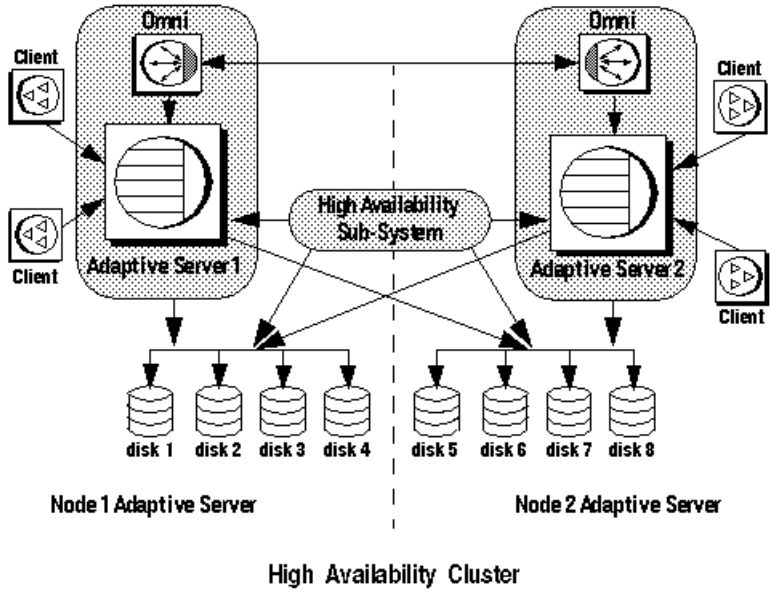
## Failover in High Availability Systems

A high availability cluster includes two or more machines that are configured so that if one machine (or application) is interrupted, the second machine assumes the workload of both machines.

Each of these machines is called one node of the high availability cluster. A high availability cluster is used in an environment that must always be available, such as a banking system to which clients must connect continuously, 365 days a year.

The machines are configured so that each machine can read the other machine's disks, although not at the same time. (All of the disks that are failed-over should be shared disks).

**Figure 1: High Availability Cluster Using Failover**




For example, if Adaptive Server 1 is the primary companion server, and it fails, Adaptive Server 2, as the secondary companion server, reads its disks (disks 1 – 4) and manages any databases on them until Adaptive Server 1 can be restarted. Any clients connected to Adaptive Server 1 are automatically connected to Adaptive Server 2.

Failover allows Adaptive Server to work in a high availability cluster in active-active or active-passive configuration.

During failover, clients connected to the primary companion using the failover property automatically reestablish their network connections to the secondary companion. Failover can be enabled by setting the connection property `HASession` to 1 (default value is "0"). If this property is not set, the session failover does not occur, even if the server is configured for failover. You must also set `SecondaryServer` (the IP address or the machine name of the secondary Adaptive Server) and `SecondaryPort` (the port number of the secondary Adaptive Server) properties. See *Adaptive Server Enterprise Using Sybase Failover in a High Availability System* for information about configuring your system for high availability.

When the Adaptive Server ODBC Driver detects a connection failure with the primary Adaptive Server, it first tries to reconnect to the primary. If it cannot reconnect, it assumes that a failover has occurred. Then, it automatically tries to connect to the secondary Adaptive Server using the connection properties set in `SecondaryServer`, and `SecondaryPort`.

### *Successful Failover Confirmation*

If a connection to the secondary server is established, the Adaptive Server ODBC Driver returns **SQL_ERROR** for the function return code. To confirm a successful failover, examine the **SQLState** and **NativeError** messages for values of 08S01 and 30130 respectively. The error message returned on such failover is:

```
"Connection to Sybase server has been lost, you have been
successfully connected to the next available HA server. All active
transactions have been rolled back."
```

You can access these values by calling **SQLGetDiagRec** on the **StatementHandle**. Then, the client must reapply the failed transaction with the new connection. If failover occurs while a transaction is open, only changes that were committed to the database before failover are retained.

### *An Unsuccessful Failover Verification*

If the connection to the secondary server is not established, the Adaptive Server ODBC Driver returns **SQL_ERROR** for the function return code.

To confirm that failover did not occur, examine the **SQLState** and **NativeError** for values of 08S01 and 30131. The error message returned on an unsuccessful failover is:

```
"Connection to Sybase server has been lost, connection to the next
available HA server also failed. All active transactions have been
rolled back".
```

You can access these values by calling **SQLGetDiagRec** on the **StatementHandle**.

To code for a failover:

```
/* Declare required variables */
....
/* Open Database connection */
....
/* Perform a transaction */
...
/* Check return code and handle failover */
if( retcode == SQL_ERROR )
{
   retcode = SQLGetDiagRec(stmt, 1,
      sqlstate,&NativeError, errmsg,100, NULL );
   if(retcode == SQL_SUCCESS ||
      retcode == SQL_SUCCESS_WITH_INFO)
   {
        if(NativeError == 30130 )
        {
        /* Successful failover retry transaction*/
        ...
        }
     else if (NativeError == 30131)
     {
        /* Failover failed. Return error */
        ...
```

```
      }
    }
}
```

## Using Failover on Microsoft Windows

Use the ODBC DataSource Administrator to failover.

1. Launch the ODBC DataSource Administrator.
2. Select the datasource you want to use and choose **Configure**.
3. Click the **Connection** tab.
4. Select **Enable High Availability** in the High Availability Information Group.
5. Provide the failover server name in the **Server Name** field.
6. Provide the failover server port in the **Server Port** field.

## Using Failover on UNIX

Use the unixODBC Driver Manager or the Sybase iAnywhere ODBC Driver Manager to failover on UNIX platform.

1. If you are linking to the unixODBC Driver Manager, edit the datasource template and reinstall the datasource using the unixODBC command line tool:

   ```
   #  odbcinst -i -s -f dsn template file
   ```

   where *dsn template file* is the complete path to the Adaptive Server ODBC datasource template file.
2. If you are directly linking to the Adaptive Server ODBC Driver or Sybase iAnywhere ODBC Driver Manager, modify the `odbc.ini` file.

   This is an example of the `odbc.ini` datasource template file:

   ```
   [sampledsn]
   Driver=Adaptive Server Enterprise
   Server=sampleserver
   Port=4100
   UserID=sa
   Password=
   Database=pubs2
   HASession=1
   SecondaryHost=failoverserver
   SecondaryPort=5000
   ```

# Kerberos Authentication

Kerberos is an industry standard network authentication system that provides simple login authentication as well as mutual login authentication.

It is used for single sign-on across various applications in extremely secure environments. Instead of passing passwords around the network, a Kerberos server holds encrypted versions of the passwords for users as well as available services.

In addition, Kerberos uses encryption to provide confidentiality and data integrity.

Adaptive Server and the Adaptive Server ODBC Driver provide support for Kerberos connections. The Adaptive Server ODBC driver specifically supports MIT, CyberSafe, and Active Directory (key distribution centers, called KDCs).

## Process Overview

An overview of the Kerberos authentication process.

1. A client application requests a ticket from the Kerberos server to access a specific service.
2. The Kerberos server returns the ticket, which contains two packets, to the client: The first packet is encrypted using the user password. The second packet is encrypted using the service password. Inside each of these packets is a session key.
3. The client decrypts the user packet to get the session key.
4. The client creates a new authentication packet and encrypts it using the session key.
5. The client sends the authentication packet and the service packet to the service.
6. The service decrypts the service packet to get the session key and decrypts the authentication packet to get the user information.
7. The service compares the user information from the authentication packet with the user information that was also contained in the service packet. If the two match, the user has been authenticated.
8. The service creates a confirmation packet that contains service specific information, as well as validation data contained in the authentication packet.
9. The service encrypts this data with the session key and returns it to the client.
10. The client uses the session key obtained from the user packet it received from Kerberos to decrypt the packet and validates that the service is what it claims to be.

In this way, the user and the service are mutually authenticated. All future communication between the client and the service (in this case, the Adaptive Server database server) will be encrypted using the session key. This successfully protects all data sent between the service and client from unwanted viewers.

## Requirements

To use Kerberos as an authentication system, you must configure Adaptive Server Enterprise to delegate authentication to Kerberos.

See the *Adaptive Server Enterprise System Administration Guide*.

If Adaptive Server has been configured to use Kerberos, any client that interacts with Adaptive Server must have a Kerberos client library installed.

This varies for operating system vendors:

- On Microsoft Windows, the Active Directory client library comes installed with the operating system.
- CyberSafe and MIT client libraries are available for Microsoft Windows and Linux.

For additional information, refer to vendor documentation.

## Enabling Kerberos Authentication

Enable Kerberos authentication by adding connection properties in the key distribution center (KDC).

1. To enable Kerberos authentication for the Adaptive Server ODBC Driver, add these connection properties:

   ```
   AuthenticationClient=<one of 'mitkerberos'
   or 'cybersafekerberos' or 'activedirectory'>
   and ServerPrincipal=<Adaptive Server name>
   ```

   where *<Adaptive Server name>* is the logical name or the principal as configured in the KDC. The Adaptive Server ODBC Driver uses this information to negotiate Kerberos authentication with the configured KDC and Adaptive Server.

   The Kerberos client libraries are compatible across various KDCs. For example, on Linux you can set `AuthenticationClient` equal to `mitkerberos`, even if your KDC is a Microsoft Active Directory.

2. If you want the Kerberos client to look for the Ticket Granting Ticket (TGT) in another cache, specify the `userprincipal` property.

   If you use **SQLDriverConnect** with the SQL_DRIVER_NOPROMPT, **ConnectString** appears as:

   ```
   "Driver=Adaptive Server Enterprise;UID=sa;
   PWD='';Server=sampleserver;
   Port=4100;Database=pubs2;
   AuthenticationClient=mitkerberos;
   ServerPrincipal=MANGO;”
   ```

### Enabling Kerberos for Login Authentication on Microsoft Windows

Enable Kerberos login authentication using the Microsoft Windows ODBC Data Source.

1. Start the Microsoft Windows ODBC Data Source administrator.
2. Select the **Sybase Adaptive Server Enterprise ODBC Driver**.
3. Select the **User DSN**/ **System DSN** tab and click the datasource that you would like to modify, or choose **Add New Data Source**.
4. On the **Security** tab, select **Use Active Directory** under the **Kerberos Authentication Client**.
5. Enter the name of the server principal in the **Server Principal** edit box. This name should match the name of the Adaptive Server configured in the KDC.

### Enabling Kerberos for Login Authentication on UNIX

Enable Kerberos login authentication using the unixODBC Driver Manager or the Sybase iAnywhere Driver Manager.

If you are linking to the unixODBC Driver Manager:

1. Open an existing datasource, or create a new datasource template.
2. Add these lines to the datasource template:

   ```
   Authentication= mitkerberos
   (or cybersafekerberos) ServerPrincipal=<MANGO>
   to enable Kerberos Login Authentication.
   ```

   where: <*MANGO*> is the name of the principal server used to authenticate sign-ons.
3. Reinstall the datasource using the **odbcinst** utility at the command line:

   ```
   odbcinst -i-s  -f ${datasourcetemplatefile}
   ```

If you are linking directly to the Adaptive Server ODBC Driver or the Sybase iAnywhere ODBC Driver Manager, modify the `odbc.ini` file directly.

This is an example of how the `odbc.ini` datasource template file should look after you modify it:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
AuthenticationClient=mitkerberos
ServerPrincipal=MANGO
```

## Obtaining an Initial Ticket From the Key Distribution Center

Generate an initial ticket called Ticket Granted Ticket (TGT) from the key distribution center (KDC) to use Kerberos authentication.

The procedure to obtain this ticket depends on the Kerberos libraries being used. For more information, refer to the vendor documentation.

### Generating TGTs for the MIT Kerberos Client Library

Review the instructions to generate TGTs for the MIT Kerberos Client Library.

1. Start the **kinit** utility at the command line:

   ```
   % kinit
   ```

2. Enter the **kinit** user name, such as your_name@YOUR.REALM.
3. Enter the password for your_name@YOUR.REALM, such as my_password. When you enter your password, the **kinit** utility submits a request to the Authentication Server for a TGT.

   The password is used to compute a key, which in turn is used to decrypt part of the response. The response contains the confirmation of the request, as well as the session key. If you entered your password correctly, you now have a TGT.
4. Verify that you have a TGT by entering this line at the command line:

   ```
   % klist
   ```

   The results of the **klist** command should be:

   ```
   Ticket cache: /var/tmp/krb5cc_1234

   Default principal: your_name@YOUR.REALM

   Valid starting      Expires             Service principal

   24-Jul-95 12:58:02  24-Jul-95 20:58:15  krbtgt/
   YOUR.REALM@YOUR.REALM
   ```

   Explanation of Results:

   - **Ticket cache –** tells you which file contains your credentials cache.
   - **Default principal –** is the login of the person who owns the TGT (in this case, you).
   - **Valid starting/Expires/Service principal –** the remainder of the output is a list of your existing tickets. Because this is the first ticket you have requested, there is only one ticket listed. The service principal (`krbtgt/YOUR.REALM@YOUR.REALM`) shows that this ticket is a TGT. Note that this ticket is good for approximately 8 hours.

# Logging Without ODBC Driver Manager Tracing

Log calls to ODBC APIs without using ODBC Driver Manager tracing in the Adaptive Server ODBC Driver.

This is useful when the driver manager is not used or when running on a platform that does not support tracing.

1. To enable this feature on Microsoft Windows, use the LOGCONFIGFILE environment variable or the Microsoft Windows registry.
2. To enable on Linux, use LOGCONFIGFILE.

   When using LOGCONFIGFILE, set the environment variable to the full path of the ODBC log's configuration file. LOGCONFIGFILE overrides any existing registry entry.

   When using the Microsoft Windows registry, create an entry called `LogConfigFile` in `HKEY_CURRENT_USER\Software\Sybase\ODBC` or `HKEY_LOCAL_MACHINE\Software\Sybase\ODBC`, and set its value to the full path of the ODBC log's configuration file. For example:

   ```
   Windows Registry Editor Version 5.00

   [HKEY_CURRENT_USER\Software\Sybase\ODBC]
   "LogConfigFile"="c:\\temp\\odbclog.properties"
   ```
3. To disable logging, delete or rename the *LogConfigFile* value.

**Note:** The value specified in `HKEY_CURRENT_USER` overrides any value set in `HKEY_LOCAL_MACHINE`.

## Log Configuration File

The configuration file controls the format and location of the ODBC log file.

This example specifies where the log file is saved:

```
log4cplus.rootLogger=OFF, NULL

log4cplus.logger.com.sybase.dataaccess.odbc.api=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.parameter=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.parameter=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.returncode=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.returncode=fals
e

log4cplus.appender.NULL=log4cplus::NullAppender
```

```
log4cplus.appender.ODBCTRACE=log4cplus::FileAppender
log4cplus.appender.ODBCTRACE.File=c:\temp\odbc.log
log4cplus.appender.ODBCTRACE.layout=log4cplus::PatternLayout
log4cplus.appender.ODBCTRACE.ImmediateFlush=true
log4cplus.appender.ODBCTRACE.layout.ConversionPattern=%d{%H:%M:%S.
%q} %t %p%-25.25c{2} %m%n
```

## Dynamic Logging Support Without ODBC Driver Manager Tracing

Starting with Adaptive Server Enterprise ODBC Driver 15.7 ESD #4, you can dynamically enable or disable the application logging during application execution by setting the **SQL_OPT_TRACE** environment attribute.

Valid values are 0 (default) to disable or 1 to enable.

```
// enable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)1,
  SQLINTEGER);
// disable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)0,
  SQLINTEGER);
```

- Dynamic logging is enabled and disabled globally and affects all connections regardless of when they were opened and whether they are part of the environment handle used to set SQL_OPT_TRACE.
- By default, the log is written to the `sybodbc.log` file in the current directory. Use the **SQL_OPT_TRACEFILE** environment attribute to set a different file/path.
- Setting the LOGCONFIGFILE environment variable or registry value enables logging for the entire duration of application execution and overrides **SQL_OPT_TRACE**.
- If an ODBC Driver Manager is being used, setting **SQL_OPT_TRACE** turns on the Driver Manager tracing and has no impact on driver tracing.
- The client application can use a null handle when linking directly against the driver or an allocated handle when using Driver Manager tracing.
- `log4cplus` configuration file cannot be used with **SQL_OPT_TRACE**.

# TDS Protocol Capture

The `ProtocolCapture` connection string captures Tabular Data Stream™ (TDS) packets exchanged between an ODBC application and the server for debugging purposes.

This property is enabled by specifying the capture file prefix.

`ProtocolCapture` takes effect immediately, so that TDS packets exchanged during connection establishment are written to a unique filename generated using the specified file prefix. TDS packets are written to the file for the duration of the connection. Use Ribo and other protocol translation tools to interpret the TDS capture files.

For example, to specify `prefix<pid>.<seqno>.tds` , example, tds_capture1280.0.tds as the TDS capture file prefix, type:

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

The first connection generates `tds_capture1280.0.tds`, the second connection generates `tds_capture1280.1.tds`, and so forth.

**Note:** Captured TDS protocol data saved to a file contains sensitive user authentication information and may contain confidential company or customer data. To protect this confidential data from unauthorized or accidental disclosure, the files containing captured data must be properly protected using file permissions or encryption.

## Dynamic Control of TDS Protocol Capture

The **SQL_ATTR_TDS_CAPTURE** connection attribute of Adaptive Server Enterprise ODBC Driver lets you pause (**SQL_CAPTURE_PAUSE**) and resume (**SQL_CAPTURE_RESUME**) the TDS protocol capture.

```
// pause protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_PAUSE, SQLINTEGER);

// resume protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_RESUME, SQLINTEGER);
```

By default, TDS protocol capture operates for the duration of the connection when the `ProtocolCapture` connection property is set for the connection. Using **SQL_ATTR_TDS_CAPTURE** (with the `ProtocolCapture` connection property) allows the application to selectively pause and resume TDS protocol capture for desired segments of program execution.

**SQL_ATTR_TDS_CAPTURE** can be set after a connection handle is allocated. It is not an error to pause or resume TDS protocol capture before a connection is established or for a connection that is not using TDS protocol capture. Pausing or resuming TDS protocol capture may be delayed by the driver to ensure the integrity of the capture stream. This ensures the write of full PDU packets for accurate capture consumption by Ribo and other protocol translator utilities.

Do not set **SQL_ATTR_TDS_CAPTURE** for applications that need to capture all TDS packets for a connection.

# ODBC Data Batching Without Binding Parameter Arrays

When the same SQL statement is executed for different parameter values, client applications normally bind parameter arrays and execute each set of parameters using **SQLExecute**, **SQLExecuteDirect**, and **SQLBulkOperations**.

In binding arrays to SQL parameters, memory for the array is allocated, and all data is copied to the array before the SQL statement is executed. This can lead to inefficient use of memory and resources when processing high volume of transactions. This behavior is seen with Adaptive Server ODBC Driver versions earlier than 15.7.

In Adaptive Server ODBC Driver 15.7 or later, client applications can use **SQLExecute** to send parameters in batches to Adaptive Server, without binding the parameters as arrays. **SQLExecute** returns SQL_BATCH_EXECUTING until the last batch of parameters has been sent and processed. It returns the status of the execution after the final batch of parameters has been processed.

A call to **SQLRowCount** is valid only after the final **SQLExecute** statement has completed.

## Managing Data Batches

Use **SQL_ATTR_BATCH_PARAMS**, a Sybase-specific connection attribute, to manage the batches of parameters sent to Adaptive Server.

Set **SQL_ATTR_BATCH_PARAMS** using **SQLSetConnectAttr**.

Valid values are:

- **SQL_BATCH_ENABLED** – informs Adaptive Server ODBC Driver to batch the parameters. When in this state, the driver sends an error if a statement other than the statement being processed—the first statement executed after setting **SQL_ATTR_BATCH_PARAMS** to **SQL_BATCH_ENABLED**—by **SQLExecute** is executed on the connection.
- **SQL_BATCH_LAST_DATA** – specifies that the next batch of parameters is the last batch, and that the parameters contain data.
- **SQL_BATCH_LAST_NO_DATA** – specifies that the next batch of parameters is the last batch, and to ignore the parameters.
- **SQL_BATCH_CANCEL** – informs the Adaptive Server ODBC Driver to cancel the batch and to roll back the transactions.
  Only uncommitted transactions can be rolled back.
- **SQL_BATCH_DISABLED** – (default) Adaptive Server ODBC Driver returns to this state after processing the last batch of parameters. You cannot manually set **SQL_ATTR_BATCH_PARAMS** to this value.

**Example1** – sends a batch of parameters to the server without binding parameter arrays:

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to start
// the batch
```

```
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
   (SQLPOINTER)SQL_BATCH_ENABLED, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Bind the parameters. This can be done once for the entire batch
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
   SQL_C_LONG, SQL_INTEGER, l1, 0, &c1, l1, &l1);
sr = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_LONGVARCHAR, l2, 0, buffer, l2, &l2);
}

// Run a batch of 10 for (int i = 0; i < 10; i++)
{
   c1 = i;
   memset(buffer, 'a'+i, l2);
   sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
   printError(sr, SQL_HANDLE_STMT, stmt);
}
```

**Example 2** – ends and closes a batch:

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to end
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
  (SQLPOINTER)SQL_BATCH_LAST_NO_DATA, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Call SQLExecDirect one more time to close the batch
// - Due to SQL_BATCH_LAST_NO_DATA, this will not
// process the parameters
sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
```

## Considerations

Considerations to manage batches.

- The parameter batching is enabled only when the `HomogeneousBatch` connection parameter is set to a value other than 0. When `HomogeneousBatch` is 2 and `EnableBulkLoad` is not 0, simple insert statements are executed using Adaptive Server bulk insert protocol. If a non-insert statement is executed or if a more complex insert statement is executed, then Adaptive Server batch protocol is used.
- This feature supports only statements and stored procedures that do not return a result set or have an output parameter.
- Asynchronous mode is not supported. While in batch mode, it is invalid for the application to execute any statement on the same connection other than the one being batched.
- **SQL_DATA_AT_EXEC** is not supported. Bind LOB parameters as normal parameters.
- When batching data without binding parameter arrays and **SQL_ATTR_PARAM_STATUS_PTR** is set, Adaptive Server ODBC Driver retrieves the number of array elements from the **StringLength** parameter to **SQLSetStmtAttr**, and not from **SQL_ATTR_PARAMSET_SIZE**.

### Bulk Insert Support for ODBC Data Batching

Starting with 15.7 ESD #4, the ODBC data batching without binding parameter arrays feature supports inserting batches using Bulk Insert protocol.

Set the `EnableBulkLoad` connection property to the desired bulk level (1, 2, or 3), and `HomogeneousBatch` connection property to 2.

For example, add `;enablebulkload=3;homogeneousbatch=2` in the connection string and simple insert statements executed in a batch are converted to fast-logged bulk insert statements.

Alternatively, set the connection properties programmatically using the **SQL_ATTR_HOMOGENEOUS_BATCH** and **SQL_ATTR_ENABLE_BULK_LOAD** connection attributes to achieve the same result:

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_HOMOGENEOUS_BATCH,
(SQLPOINTER)2, SQL_IS_INTEGER);
sr = SQLSetConnectAttr(hdbc,
SQL_ATTR_ENABLE_BULK_LOAD, (SQLPOINTER)3, SQL_IS_INTEGER);
```

## ODBC Deferred Array Binding

Adaptive Server Enterprise ODBC Driver provides the extended `SQLBindColumnDA()` and `SQLBindParameterDA()` APIs that allow applications to bind all columns or parameters with a single API call.

When you use these APIs, the pointers to column buffer or parameter buffer are reevaluated for each `SQLExecute()` or `SQLExecDirect()` call. Therefore, the application is able to change the buffers without another `SQLBindCol()` or `SQLBindParameter()` call. Because the calls to bind new pointers can be expensive, using the new extended APIs improves application performance where the same statement needs to be executed many times. Applications may also be able to save some memory copy operations by changing the buffer pointers before executing a query such that data is read from where available or copied to where needed.

### SQLBindColumnDA()

Binds a buffer to a set of column markers.

*Syntax*

```
SQLRETURN SQLBindColumnDA(
   SQLHSTMT StatementHandle,
   SQLSMALLINT* TargetTypes,
   SQLSMALLINT* Precisions,
   SQLSMALLINT* Scales,
   SQLPOINTER* TargetValuePtrs,
   SQLLEN* BufferLengths,
```

```
   SQLLEN** StrLens_or_Inds,
   SQLUSMALLINT Columns)
```

*Parameters*

**StatementHandle** – [Input] Statement handle.

**TargetTypes** – [Input] The C types of *TargetValuePtrs*. A copy of the array is made. The only way to update the C type of a column is to call this function again.

**Precisions** – [Deferred Input] The precision to use for this column buffer.

**Scales** – [Deferred Input] The scale to use for this column buffer.

**TargetValuePtrs** – [Deferred Input/Output] Pointers to the data buffers to bind to the columns. The elements of the array must be non-NULL.

**BufferLengths** – [Deferred Input] Length of the **TargetValuePtrs** buffers in bytes.

**StrLens_or_Inds** – [Deferred Input/Output] Pointer to the length/indicator buffers to bind to the columns.

**Columns** – [Input] The number of columns bound.

## SQLBindParameterDA()

Binds a buffer to a set of parameter markers.

*Syntax*

```
SQLRETURN SQLBindParameterDA(
   SQLHSTMT StatementHandle,
   SQLSMALLINT* InputOutputTypes,
   SQLSMALLINT* ValueTypes,
   SQLSMALLINT* ParameterTypes,
   SQLULEN* ColumnSizes,
   SQLSMALLINT* DecimalDigits,
   SQLPOINTER* ParameterValuePtrs,
   SQLLEN* BufferLength,
   SQLLEN** StrLens_or_IndPtrs,
   SQLUSMALLINT Parameters)
```

*Parameters*

**StatementHandle** – [Input] Statement handle.

**InputOutputTypes** – [Input] The types of the parameters. A copy of the array is made. The only way to update the **InputOutputType** of a parameter is to call this function again.

**ValueTypes** – [Input] The C datatypes of the parameters. A copy of the array is made. The only way to update the **ValueType** of a parameter is to call this function again.

**ParameterTypes** – [Input] The SQL datatypes of the parameters. A copy of the array is made. The only way to update the **ParameterType** of a parameter is to call this function again.

**ColumnSizes** – [Input] The size of the columns or expressions of the corresponding parameter markers. A copy of the array is made. The only way to update the **ColumnSize** of a parameter is to call this function again.

**DecimalDigits** – [Input] The decimal digits of the column or expression of the corresponding parameter markers. A copy of the array is made. The only way to update the **DecimalDigits** of a parameter is to call this function again.

**ParameterValuePtrs** – [Deferred Input/Output] An array of pointers to the buffers for the parameters' data. The elements of the array must be non-NULL.

**BufferLength** – [Deferred Input] An array of buffer lengths.

**StrLens_or_IndPtrs** – [Deferred Input] An array of pointers to the buffers for the parameters' length.

**Parameters** – [Input] The number of parameters bound.

## Usage

The usage instructions for `SQLBingColumnDA()` and `SQLBindParameter()` APIs.

- You cannot mix standard bindings and deferred array bindings.
- All bindings must be cleared out when switching between standard bindings and deferred array bindings.
- When using deferred array bindings, all columns or parameters must be bound in order without omission using a single API call because a subsequent call to `SQLBindColumnDA()` or `SQLBindParameterDA()` replaces the values from the previous call.
- The `SQLExecute()` and `SQLExecDirect()` functions can now return the errors of `SQLBindParameter()` related to *BufferLength* if `SQLBindParameterDA()` is used to bind the parameters.
- The `SQLFetch()` function can now return the errors of `SQLBindCol()` related to *BufferLength* if `SQLBindColumnDA()` is used to bind the columns.
- `SQLBindColumnDA()` and `SQLBindParameterDA()` cannot be used with ODBC Driver Managers since the feature utilizes a non-standard API call.
- `SQLSetDescField()` usage limitations: Some values of *FieldIdentifier* are not allowed when deferred array bindings are used. For example, **SQL_DESC_DATA_PTR** returns an error because the application should change the *ValuePtr* array to change the application buffer used. Any *FieldIdentifier* that would update an `SQLBindCol()` or `SQLBindParameter()` field would return an error when deferred array bindings are used.

**Sample Program**

The **dabinding** sample program demonstrates this feature.

## Suppressing Additional Row Format Information

Use the `SuppressRowFormat2` connection string property to force Adaptive Server to send data using the `TDS_ROWFMT` byte sequence where possible instead of the `TDS_ROWFMT2` byte sequence.

`TDS_ROWFMT` contains less data than `TDS_ROWFMT2`, which includes catalog, schema, table, and column information and can result in better performance for many small **select** operations. Because the server sends reduced result set metadata when `SuppressRowFormat2` is set to 1, some information is not available to client programs. If your application relies on the missing metadata, you should not enable this property.

Values:

- 0 – the default value; `TDS_ROWFMT2` is not suppressed.
- 1– forces the server to send data in `TDS_ROWFMT` where possible.

**Note:** You should not use the `SuppressRowFormat2` connection string property with an ODBC program that uses the `SQLBulkOperations` API. Enabling `SuppressRowFormat2` suppresses information that SQL bulk operations requires and will result in an error.

When you are connecting to Adaptive Server 15.7 ESD #1 or later, the `SuppressRowFormat2` property should be considered obsolete. Use the `SuppressRowFormat` connection property instead for better performance results and lesser restrictions.

## Suppressing Row Format Metadata

Improve the performance of repeatedly executed queries with the Adaptive Server ODBC driver by instructing Adaptive Server to suppress row format metadata (TDS_ROWFMT or TDS_ROWFMT2) for queries that are re-executed in a session. Adaptive Server 15.7 ESD#1 and later supports row format metadata suppression.

To suppress row format metadata, use the `SuppressRowFormat` connection string property.

The valid `SuppressRowFormat` connection string property values are:

- 0 – row format metadata is not suppressed.
- 1– the default value; Adaptive Server will not send row format metadata where possible.

**Note:** You can suppress row format metadata only if the connected Adaptive Server supports this feature. If the `SuppressRowFormat` is 1 but the connected Adaptive Server does not support the suppression of row format metadata, Adaptive Server ignores the parameter.

For example, this ODBC connection string causes row format metadata to be suppressed:

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressRowFormat=1;
```

## Suppressing Parameter Format Metadata

Improve the performance of prepared statements with the ODBC driver by suppressing parameter format metadata when the prepared statements are re-executed.

Adaptive Server 15.7 ESD#1 and later supports parameter format metadata suppression.

To suppress parameter format metadata, set the `DynamicPrepare` connection property to 1, and then use the `SuppressParamFormat` connection string property.

The valid `SuppressParamFormat` connection string property values are:

- 0 – parameter format metadata is not suppressed in prepared statements.
- 1– the default value; parameter format metadata is suppressed where possible.

**Note:** You can suppress parameter format metadata in prepared statements only if the connected Adaptive Server supports this feature. If the `DynamicPrepare` and `SuppressParamFormat` parameters are both set to 1 but the connected Adaptive Server does not support the suppression of parameter format metadata, Adaptive Server ignores the parameter settings.

For example, this ODBC connection string causes parameter format metadata to be suppressed in prepared statements:

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;SuppressParam
Format=1;
```

## Releasing Locks at Cursor Close

Include the **release_locks_on_close** option in the **declare cursor** syntax , which releases shared cursor locks at isolation levels 2 and 3 when a cursor is closed. Adaptive Server ODBC Driver accordingly supports the release-lock-on-close semantics.

To apply this functionality to all read-only cursors created on an Adaptive Server ODBC Driver connection, set the `ReleaseLocksOnCursorClose` connection property to 1.

The default `ReleaseLocksOnCursorClose` value is 0.

Settings applied through the `ReleaseLocksOnCursorClose` connection property is static and cannot be changed after the connection has been established. This setting takes effect only when connected to a server that supports **release_locks_on_close**.

See **release_locks_on_close** in the Adaptive Server Enterprise *Reference Manual: Commands*.

## select for update Support

Adaptive Server supports **select for update**, which can lock rows for subsequent updates within the same transaction, and supports exclusive locks for updatable cursors.

See *Queries: Selecting Data from a Table* in the Adaptive Server Enterprise *Transact-SQL Users Guide*.

This functionality is automatically available to clients when the **for update** clause is added to a **select** statement and to any updatable cursors opened within the clients.

## Variable-Length Rows in Data-Only Locked Tables

Versions of Adaptive Server earlier than 15.7 configured for 16K logical page sizes could not create data-only locked (DOL) tables with variable-length rows if a variable-length column began more than 8191 bytes after the start of the row.

This limitation has been removed starting in Adaptive Server 15.7. See *Data Storage* in the Adaptive Server Enterprise *Performance and Tuning Series: Physical Database Tuning*.

ODBC clients do not need special configuration to use this feature. When connected to Adaptive Server version 15.7 that has been configured to receive wide **DOL** rows, these clients automatically insert records using the wide offset. An error message is received if a client attempts to send a wide **DOL** row to an earlier version of Adaptive Server, or to a 15.7 Adaptive Server for which the wide **DOL** row option is disabled.

## Nonmaterialized Columns

The bulk insert routines in the Adaptive Server ODBC Driver version 15.7 and later can handle nonmaterialized columns in Adaptive Server 15.7.

Earlier versions of the Adaptive Server ODBC Driver cannot perform bulk inserts of data into Adaptive Server when a table definition contains nonmaterialized columns. Adaptive Server will raise an error when earlier versions of the Adaptive Server ODBC Driver attempt to perform bulk inserts into nonmaterialized columns.

# Large Object (LOB) Support

Adaptive Server ODBC Driver supports Large Object (LOB) datatypes.

The LOB datatypes `text`, `unitext`, and `image` are supported as:

- LOB columns with in-row storage
  In Adaptive Server, LOB columns that are marked for in-row are stored in-row when there is adequate memory to hold the entire row. When the size of a row increases over its defined limit due to an update to any column in it, the LOB columns which are stored in-row are moved off-row to bring it within the limits. See *In-Row Off-Row LOB* in the Adaptive Server Enterprise *Transact-SQL Users Guide*.
  The bulk insert routines in Adaptive Server ODBC Driver support the in-row and off-row storage of `text`, `image`, and `unitext` LOB columns in Adaptive Server. Bulk insert routines from earlier client versions always store LOB columns off row.
- LOB objects as parameters of stored procedures
  Adaptive Server ODBC Driver supports using `text`, `unitext`, and `image` as input parameters in stored procedures and as parameter marker datatypes.

# Large Object (LOB) Locator Support

Adaptive Server ODBC Driver supports large object (LOB) locators. A LOB locator contains a logical pointer to LOB data rather than the data itself, reducing the amount of data that passes through the network between Adaptive Server and its clients.

Adaptive Server ODBC Driver clients cannot use LOB locators unless connected to an Adaptive Server that supports it. Adaptive Server has server support for LOB locators.

**Note:** When you are using LOB locators, retrieving a large result set that includes LOB data on each row may impact your application's performance. Adaptive Server returns a LOB locator as part of the result set and, to obtain LOB data, Adaptive Server ODBC Driver must cache the remaining result set. Sybase recommends that you keep result sets small, or that you enable cursor support to limit the size of data to be cached.

## Enabling LOB Locator Support

Use the `EnableLOBLocator` connection property to enable the LOB locator support.

1. To enable LOB locator support in Adaptive Server ODBC Driver, establish a connection to Adaptive Server with the `EnableLOBLocator` connection property set to 1.

When `EnableLOBLocator` is 0, the default value, the Adaptive Server ODBC Driver cannot retrieve a locator for a LOB column. When enabling LOB Locators, the connection should be set to **autocommit off**.

2. Include the `sybasesqltypes.h` file in your program.

   The `sybasesqltypes.h` file is located in the `include` directory, under the ODBC installation directory.

**Note:** LOB Locators can be used reliably when the application is linked directly to the driver. When a driver manager is used and LOB Locators are used, some driver managers may restrict the use of vendor defined C and SQL Types and the application may encounter an Unsupported Type error.

### ODBC Datatype Mapping for Locator Support

The ODBC datatype mapping for the Adaptive Server locator datatypes.

| Adaptive Server Datatype | ODBC SQL Type | ODBC C Type |
|---|---|---|
| `text_locator` | `SQL_TEXT_LOCATOR` | `SQL_C_TEXT_LOCATOR` |
| `image_locator` | `SQL_IMAGE_LOCATOR` | `SQL_C_ IMAGE_LOCATOR` |
| `unitext_locator` | `SQL_UNITEXT_LOCATOR` | `SQL_C_ UNITEXT_LOCATOR` |

### Supported Conversions

The supported conversions for the Adaptive Server locator datatypes.

| | SQL_C_TEXT_LOCATOR | SQL_C_IMAGE_LOCATOR | SQL_C_UNITEXT_LOCATOR |
|---|---|---|---|
| `SQL_TEXT_LOCATOR` | X | | |
| `SQL_IMAGE_LOCATOR` | | X | |
| `SQL_UNITEXT_LOCATOR` | | | X |
| `SQL_LONGVARCHAR` | | | |
| `SQL_WLONGVARCHAR` | | | |
| `SQL_LONGVARBINARY` | | | |
| LEGEND: x = supported conversion. | | | |

### ODBC API Methods that Support LOB Locators

Review the Adaptive Server ODBC API methods that support LOB locators.

- `SQLBindCol` – **TargetType** can be any of the ODBC C locator datatypes.
- `SQLBindParameter` – **ValueType** can be any of the ODBC C locator datatypes. *ParameterType* can be any of the ODBC SQL locator datatypes.
- `SQLGetData` – **TargetType** can be any of the ODBC C locator datatype.
- `SQLColAttribute` – the SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE descriptors can return any of the `ODBC SQL` locator datatype.
- `SQLDescribeCol` – the datatype pointer can be any of the `ODBC SQL` locator datatypes.

See *Microsoft ODBC API Reference*.

### Retrieving Implicit Conversion of Prefetched LOB Data

Use `SQLGetData` and `SQLBindCol` to retrieve the underlying prefetched LOB data by binding the column to `SQL_C_CHAR` or `SQL_C_WCHAR` for `text` locators, or to `SQL_C_BINARY` for `image` locators, when Adaptive Server returns a LOB locator.

1. Set the **SQL_ATTR_LOBLOCATOR** attribute to enable or disable locators in a connection.

   If `EnableLOBLocator` has been specified in the connection string, **SQL_ATTR_LOBLOCATOR** is initialized with the value of `EnableLOBLocator`, otherwise, it is set to SQL_LOBLOCATOR_OFF, the default value.
2. To enable locators, set the attribute to SQL_LOBLOCATOR_ON.
3. Use **SQLSetConnectAttr** to set the attribute's value and **SQLGetConnectAttr** to retrieve its value.
4. Use **SQLSetStatementAttr** to set **SQL_ATTR_LOBLOCATOR_FETCHSIZE** to specify the size in bytes for binary data and in characters for character data of the LOB data to retrieve.

   The default value, 0, indicates that prefetched data is not requested, while a value of -1 retrieves the entire LOB data.

**Note:** If the underlying LOB data size of the column being retrieved exceeds the prefetched data size that you have set, native error 3202 is raised when an ODBC client attempts to directly retrieve the data. When this happens, the client can retrieve the complete data by calling `SQLGetData` to obtain the underlying locator and perform all of the operations available on locators.

**Example 1** – retrieves an `image` locator using `SQLGetData` when the prefetched data represents the complete LOB value:

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);
```

```
//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_ON,
   0);

// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt, SQL_ATTR_LOBLOCATOR_FETCHSIZE,
(SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);

printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);

sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

//Retrieve the binary data  (Complete Data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);
printError(sr, SQL_HANDLE_STMT, stmt);


//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

**Example 2** – retrieves an `image` locator using **SQLGetData** when prefetched data represents a truncated LOB value:

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
```

```
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
   (SQLPOINTER)SQL_LOCATOR_ON, 0);

//Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt,
   SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);
printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

// Retrieve the binary data(Truncated data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);

if(sr == SQL_SUCCESS_WITH_INFO)
{
   SQLTCHAR errormsg[ERR_MSG_LEN];
   SQLTCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER nativeerror = 0;
   SQLSMALLINT errormsglen = 0;

   retcode = SQLGetDiagRec(handleType, handle, 1, sqlstate,
&nativeerror,
      errormsg, ERR_MSG_LEN, &errormsglen);

   printf("SqlState: %s Error Message: %s\n", sqlstate, errormsg);

   //Handle truncation of LOB data; if data was truncated call
SQLGetData to
   // retrieve the locator.

   /* Warning returns truncated LOB data */
   if (NativeError == 32028) //Error code may change
   {
      BYTE ImageLocator[SQL_LOCATOR_SIZE];
      sr = SQLGetData(stmt, 1, SQL_C_IMAGE_LOCATOR, &ImageLocator,
```

```
         sizeof(ImageLocator), &Len);
      printError(sr, SQL_HANDLE_STMT, stmt);

      /*
         Perform locator specific calls using image Locator on a
separate
         statement handle if needed
      */
   }
}

//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

### Access and Manipulate LOBs Using Locators

The ODBC API does not directly support LOB locators. An ODBC client application must use Transact-SQL® functions to operate on the locators and manipulate LOB values.

Adaptive Server ODBC Driver introduces several stored procedures to facilitate the use of the required Transact-SQL functions.

The input and output values of the parameters can be of any type that Adaptive Server can implicitly convert to the stored procedure definitions.

For details about the Transact-SQL commands and functions listed here, see *Transact-SQL Functions* in the Adaptive Server Enterprise *Reference Manual: Building Blocks*.

### Initialize a Text Locator

Use **sp_drv_create_text_locator** to create a `text_locator` and optionally initialize it with value. **sp_drv_create_text_locator** accesses the Transact-SQL function **create_locator**.

#### *Syntax*

```
sp_drv_create_text_locator [init_value]
```

#### *Input Parameters*

**init_value** – a `varchar` or `text` value used to initialize the new locator.

#### *Output Parameters*

None.

*Result Set*
A column of type `text_locator`. The LOB that the locator references has **init_value** when supplied.

### Initialize a Unitext Locator

Use **sp_drv_create_unitext_locator** to create a `unitext_locator` and optionally initialize it with value. **sp_drv_create_unitext_locator** accesses the Transact-SQL function **create_locator**.

*Syntax*

```
sp_drv_create_unitext_locator [init_value]
```

*Input Parameters*
**init_value** – a `univarchar` or `unitext` used to initialize the new locator.

*Output Parameters*
None.

*Result Set*
A column of type `unitext_locator`. The LOB that the locator references has **init_value** when supplied.

### Initialize an Image Locator

Use **sp_drv_create_image_locator** to create an `image_locator` and optionally initialize it with value. **sp_drv_create_image_locator** accesses the Transact-SQL function **create_locator**.

*Syntax*

```
sp_drv_create_image_locator [init_value]
```

*Input Parameters*
**init_value** – a `varbinary` or `image` used to initialize the new locator.

*Output Parameters*
None.

*Result Set*
A column of type `image_locator`. The LOB that the locator references has **init_value** when supplied.

### Obtain Complete Text Value From a Text Locator

Use **sp_drv_locator_to_text** which accesses the Transact-SQL function **return_lob**.

*Syntax*

```
sp_drv_locator_to_text locator
```

*Input Parameters*

**locator** – `text_locator` to retrieve value of.

*Output Parameters*

None.

*Result Set*

A column containing the `text` value referenced by **locator**.

### Obtain Complete Unitext Value From a Unitext Locator

Use **sp_drv_locator_to_unitext** which accesses the Transact-SQL function **return_lob**.

*Syntax*

```
sp_drv_locator_to_unitext locator
```

*Input Parameters*

**locator** – `unitext_locator` to retrieve value of.

*Output Parameters*

None.

*Result Set*

A column containing the `unitext` value referenced by **locator**.

### Obtain Complete Image Value From an Image Locator

Use **sp_drv_locator_to_image** which accesses the Transact-SQL function **return_lob**.

*Syntax*

```
sp_drv_locator_to_image locator
```

*Input Parameters*

**locator** – `image_locator` to retrieve value of.

*Output Parameters*

None.

*Result Set*
A column containing the `image` value referenced by **locator**.

### Obtain a Substring From a Text Locator

Use **sp_drv_text_substring** which accesses the Transact-SQL function **substring**.

*Syntax*

```
sp_drv_text_substring locator, start_position, length
```

*Input Parameters*

- **locator** – a `text_locator` that references the data to manipulate.
- **start_position** – an `integer` specifying the position of the first character to read and retrieve.
- **length** – an `integer` specifying the number of characters to read.

*Output Parameters*
None.

*Result Set*
A column of type `text` containing the substring retrieved.

### Obtain a Substring From a Unitext Locator

Use **sp_drv_unitext_substring** which accesses the Transact-SQL function **substring**.

*Syntax*

```
sp_drv_unitext_substring locator, start_position, length
```

*Input Parameters*

- **locator** – a `unitext_locator` that references the data to manipulate.
- **start_position** – an `integer` specifying the position of the first character to read and retrieve.
- **length** – an `integer` specifying the number of characters to read.

*Output Parameters*
None.

*Result Set*
A column of type `unitext` containing the substring retrieved.

### Obtain a Substring From an Image Locator

Use **sp_drv_image_substring**, which accesses the Transact-SQL function **substring**.

*Syntax*

```
sp_drv_image_substring locator, start_position, length
```

*Input Parameters*

- **locator** – an `image_locator` that references the data to manipulate.
- **start_position** – an `integer` specifying the position of the first byte to read and retrieve.
- **length** – an `integer` specifying the number of bytes to read.

*Output Parameters*

None.

*Result Set*

A column of type `image` containing the substring retrieved.

### Insert Text at Specified Position

Use **sp_drv_text_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_text_setdata locator, offset, new_data, data_length
```

*Input Parameters*

- **locator** – a `text_locator` that references the `text` column to insert into.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data** – `varchar` or `text` data to insert.

*Output Parameters*

**data_length** – an `integer` containing the number of characters written.

*Result Set*

None.

### Insert Unitext at Specified Position

Use **sp_drv_unitext_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_unitext_setdata locator, offset, new_data, data_length
```

*Input Parameters*

- **locator** –a `unitext_locator` that references the `unitext` column to insert into.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data** – `univarchar` or `unitext` data to insert.

*Output Parameters*
**data_length** – an `integer` containing the number of characters written.

*Result Set*
None.

### Insert an Image at Specified Position

Use **sp_drv_image_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_image_setdata locator, offset, new_data, datalength
```

*Input Parameters*

- **locator** – an `image_locator` that references the `image` column to insert in.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data** – `varbinary` or `image` data to insert.

*Output Parameters*
**data_length** – an `integer` containing the number of bytes written.

*Result Set*
None.

### Insert Text Referenced by a Locator

Use **sp_drv_text_locator_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_text_locator_setdata locator, offset, new_data_locator,
data_length
```

*Input Parameters*

- **locator** – a `text_locator` that references the `text` column to insert into.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data_locator** – a `text_locator` that references the `text` data to insert.

*Output Parameters*
**data_length** – an `integer` containing the number of characters written.

*Result Set*
None.

### Insert Unitext Referenced by a Locator

Use **sp_drv_unitext_locator_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_unitext_locator_setdata locator, offset, new_data_locator,
data_length
```

*Input Parameters*

- **locator** – a `unitext_locator` that references the `unitext` column to insert into.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data_locator** – a `unitext_locator` that references the `unitext` data to insert.

*Output Parameters*
**data_length** – an `integer` containing the number of characters written.

*Result Set*
None.

### Insert Image Referenced by a Locator

Use **sp_drv_image_locator_setdata** which accesses the Transact-SQL function **setadata**.

*Syntax*

```
sp_drv_image_locator_setdata locator, offset, new_data_locator,
   datalength
```

*Input Parameters*

- **locator** – an `image_locator` that references the `image` column to insert in.
- **offset** – an `integer` specifying the position from which to start writing the new content.
- **new_data_locator** – an `image_locator` that references the `image` data to insert.

*Output Parameters*
**data_length** – an `integer` containing the number of bytes written.

*Result Set*
None.

### Truncate Underlying LOB Data

Use **truncate lob** to truncate the LOB data referenced by a LOB locator.

See the Adaptive Server Enterprise *Reference Manual: Commands*.

### Find Character Length of Underlying Text Data

Use **sp_drv_text_locator_charlength** to find the character length of a LOB column referenced by a `text` locator. **sp_drv_text_locator_charlength** accesses the Transact-SQL function **char_length**.

*Syntax*

```
sp_drv_text_locator_charlength locator, data_length
```

*Input Parameters*

**locator** – a `text_locator` that references the `text` column to manipulate.

*Output Parameters*

**data_length** – an `integer` specifying the character length of the `text` column referenced by **locator**.

*Result Set*

None.

### Find Byte Length of Underlying Text Data

Use **sp_drv_text_locator_bytelength** to find the byte length of a LOB column referenced by a `text` locator. **sp_drv_text_locator_bytelength** accesses the Transact-SQL function **data_length**.

*Syntax*

```
sp_drv_image_locator_bytelength locator, data_length
```

*Input Parameters*

**locator** – a `text_locator` that references the `text` column to manipulate.

*Output Parameters*

**data_length** – an `integer` specifying the byte length of the `text` column referenced by **locator**.

*Result Set*

None.

### Find Character Length of Underlying Unitext Data

Use **sp_drv_unitext_locator_charlength** to find the character length of a LOB column referenced by a `unitext` locator. **sp_drv_unitext_locator_charlength** accesses the Transact-SQL function **char_length**.

*Syntax*

```
sp_drv_unitext_locator_charlength locator, data_length
```

*Input Parameters*

**locator** – a `unitext_locator` that references the `unitext` column to manipulate.

*Output Parameters*

**data_length** – an `integer` specifying the character length of the `unitext` column referenced by **locator**.

*Result Set*

None.

### Find Byte Length of Underlying Unitext Data

Use **sp_drv_unitext_locator_bytelength** to find the byte length of a LOB column referenced by a `unitext` locator. **sp_drv_unitext_locator_bytelength** accesses the Transact-SQL function **data_length**.

*Syntax*

```
sp_drv_image_locator_bytelength locator, data_length
```

*Input Parameters*

**locator** – a `unitext_locator` that references the `unitext` column to manipulate.

*Output Parameters*

**data_length** – an `integer` specifying the byte length of the `unitext` column referenced by **locator**.

*Result Set*

None.

### Find Byte Length of Underlying Image Data

Use **sp_drv_image_locator_bytelength** to finds the byte length of a LOB column referenced by an `image` locator. **sp_drv_image_locator_bytelength** accesses the Transact-SQL function **data_length**.

*Syntax*

```
sp_drv_image_locator_bytelength locator, data_length
```

*Input Parameters*

**locator** – an `image_locator` that references the `image` column to manipulate.

*Output Parameters*

**data_length** – an `integer` specifying the byte length of the `image` column referenced by **locator**.

*Result Set*

None.

### Find Position of a Search String Within the Text Column Referenced by a Locator

Use **sp_drv_varchar_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_varchar_charindex search_string, locator, start, position
```

*Input Parameters*

- **search_string** – the literal, of type `varchar`, to search for.
- **locator** – a `text_locator` that references the `text` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**position** – an `integer` specifying the starting position of **search_string** in the LOB column referenced by **locator**.

*Result Set*

None.

### Find Position of a String Referenced by a Text Locator Within the Text Column Referenced by Another Locator

Use **sp_drv_textlocator_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_textlocator_charindex search_locator, locator, start,
position
```

*Input Parameters*

- **search_locator** – a `text_locator` that points to the literal to search for.
- **locator** – a `text_locator` that references the `text` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**position** – an `integer` specifying the starting position of the literal in the LOB column referenced by **locator**.

*Result Set*

None.

### Find Position of a Search String Within the Unitext Column Referenced by a Locator

Use **sp_drv_univarchar_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_univarchar_charindex search_string, locator, start, position
```

*Input Parameters*

- **search_string** – the literal, of type `univarchar`, to search for.
- **locator** – a `unitext_locator` that references the `unitext` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**position** – an `integer` specifying the starting position of **search_string** in the LOB column referenced by **locator**.

*Result Set*

None.

### Find Position of a String Referenced by a Unitext Locator Within the Unitext Column Referenced by Another Locator

Use **sp_drv_unitext_locator_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_charindex_unitextloc_in_locator search_locator, locator,
start,
position
```

*Input Parameters*

- **search_locator** – a `unitext_locator` that points to the literal to search for.
- **locator** – a `unitext_locator` that references the `text` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**position** – an `integer` specifying the starting position of the literal in the LOB column referenced by **locator**.

*Result Set*

None.

### Find Position of a Byte Sequence Within the Column Referenced by an Image Locator

Use **sp_drv_varbinary_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_varbinary_charindex byte_sequence, locator, start, position
```

*Input Parameters*

- **byte_sequence** – the byte sequence, of type `varbinary`, to search for.
- **locator** – an `image_locator` that references the `image` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**position** – an `integer` specifying the starting position of **search_string** in the LOB column referenced by **locator**.

*Result Set*

None.

### Find Position of Byte Sequence Referenced by an Image Locator Within the Image Column Referenced by Another Locator

Use **sp_drv_image_locator_charindex** which accesses the Transact-SQL function **charindex**.

*Syntax*

```
sp_drv_image_locator_charindex sequence_locator, locator, start,
start_position
```

*Input Parameters*

- **sequence_locator** – an `image_locator` that points to the byte sequence to search for.
- **locator** – an `image_locator` that references the `image` column to search from.
- **start** – an integer specifying the position from which to begin searching. The first position is 1.

*Output Parameters*

**start_position** – an `integer` specifying the starting position of the byte sequence in the LOB column referenced by **locator**.

*Result Set*

None.

### Check If a text_locator Reference is Valid

Use **sp_drv_text_locator_valid** which accesses **locator_valid**.

*Syntax*

```
sp_drv_text_locator_valid locator
```

*Input Parameters*

**locator** – the `text_locator` to validate.

*Output Parameters*

A `bit` representing one of these values:

- 0 – false, **locator** is invalid.
- 1 – true, **locator** is valid.

*Result Set*

None.

### Check If a unitext_locator Reference is Valid

Use **sp_drv_unitext_locator_valid** which accesses **locator_valid**.

*Syntax*

```
sp_drv_unitext_locator_valid locator
```

*Parameters*

**locator** – the `unitext_locator` to validate.

*Output Parameters*

A `bit` representing one of these values:

- 0 – false, **locator** is invalid.
- 1 – true, **locator** is valid.

*Result Set*

None.

### Check If an image_locator Reference is Valid

Use **sp_drv_image_locator_valid** which accesses **locator_valid**.

*Syntax*

```
sp_drv_image_locator_valid locator
```

*Parameters*

**locator** – the `image_locator` to validate.

*Output Parameters*

A `bit` representing one of these values:

- 0 – false, **locator** is invalid.
- 1 – true, **locator** is valid.

*Result Set*

None.

### Free or Deallocate a LOB Locator

Use **deallocate locator**.

See the Adaptive Server Enterprise *Reference Manual: Commands*.

### Examples

Some examples for enabling LOB locator support.

**Example 1** – allocates handles and establishes a connection:

```
// Assumes that DSN has been named "sampledsn" and
// UseLobLocator has been set to 1.

SQLHENV environmentHandle = SQL_NULL_HANDLE;
SQLHDBC connectionHande = SQL_NULL_HANDLE;
SQLHSTMT statementHandle = SQL_NULL_HANDLE;
SQLRETURN ret;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &environmentHandle);
SQLSetEnvAttr(environmentHandle, SQL_ATTR_ODBC_VERSION,
SQL_ATTR_OV_ODBC3);
SQLAllocHandle(SQL_HANDLE_DBC, environmentHandle,
&connectionHandle);
Ret = SQLConnect(connectionHandle, "sampledsn",
   SQL_NTS, "sa", SQL_NTS, "Sybase",SQL_NTS);
```

**Example 2** – selects a column and retrieves a locator:

```
// Selects and retrieves a locator for bk_desc, where
// bk_desc is a column of type text defined in a table
// named books. bk_desc contains the text "A book".

SQLPrepare(statementHandle, "SELECT bk_desc FROM books
   WHERE bk_id =1", SQL_NTS);

SQLExecute(statementHandle);
BYTE TextLocator[SQL_LOCATOR_SIZE];
SQLLEN Len = 0;
ret = SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);

If(Len == sizeof(TextLocator))
{
   Cout << Locator was created with expected size <<
   Len;
}
```

**Example 3** – determines data length:

```
SQLLEN LocatorLen = sizeof(TextLocator);
ret = SQLBindParameter(statementHandle, 1,
   SQL_PARAM_INPUT, SQL_C_TEXT_LOCATOR,
   SQL_TEXT_LOCATOR, SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), &LocatorLen);

SQLLEN CharLenSize = 0;
SQLINTEGER CharLen = 0;
ret = SQLBindParameter(statementHandle, 2,
SQL_PARAM_OUTPUT, SQL_C_LONG,SQL_INTEGER,0 , 0,
&CharLen, sizeof(CharLen), &CharLenSize);
SQLExecDirect(statementHandle,
   "{CALL sp_drv_text_locator_charlength( ?,?) }" , SQL_NTS);

cout<< "Character Length of Data " << charLen;
```

**Example 4** – appends text to a LOB column:

```
SQLINTEGER retVal = 0;
SQLLEN Col1Len = sizeof(retVal);
SQLCHAR appendText[10]=”abcdefghi on C++”;

SQLBindParameter(statementHandle, 14,
   SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &retVal, 0,
Col1Len);

SQLBindParameter(statementHandle, 21, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, &TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 32, SQL_PARAM_INPUT,
   SQL_C_SLONG, SQL_INTEGER, 0, 0, &charLen, 0, SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 43, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 10, 0, append_text,
   sizeof(append_text), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle,
   "{? = CALL sp_drv_setdata_text (?, ?, ?,?) }" , SQL_NTS);

SQLFreeStmt(statementHandle, SQL_CLOSE);
```

**Example 5** – retrieves LOB data from a LOB locator.

```
SQLCHAR description[512];
SQLLEN descriptionLength = 512;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle, "{CALL sp_drv_locator_to_text(?)}",
SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, 1,SQL_C_CHAR, description,
   descriptionLength, &descriptionLength)

Cout << "LOB data referenced by locator:" << description
   << endl;

Cout << "Expected LOB data:A book on C++" << endl;
```

**Example 6** – transfers data from a client application to a LOB locator.

```
description = "A lot of data that will be used for a lot
   of inserts, updates and deletes"; descriptionLength = SQL_NTS;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 512, 0, description,
   sizeof(description), &descriptionLength);
```

```
SQLExecDirect(statementHandle,
   "{CALL sp_drv_create_text_locator(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);
```

## Server-Specified Packet Size

Clients and servers have to be prepared to reserve memory to store the packages used for communication between them.

These packages are called Protocol Data Units, or PDUs. Every PDU starts with an 8-byte header containing a 2-byte, unsigned integer describing the actual size in bytes of the current PDU (including the header itself). Clients and servers must know the maximum size that a PDU sent by the other party could be, and this is called the *packet size*. The packet size is negotiated at login time.

When connected to Adaptive Server 15.0 and later, the Adaptive Server ODBC Driver lets the server select the packet size to optimize performance. When connected to an earlier version of Adaptive Server, the Adaptive Server ODBC Driver uses 512 as the packet size, unless you specify the `packetsize` property. If you do not want the server to decide the packet size, you need to set `EnableServerPacketSize` to 0. If you have memory restrictions, set `RestrictMaximumPacketSize` to a number (in multiples of 512) so that Adaptive Server and the Adaptive Server ODBC Driver do not negotiate a packet size greater than the one you specified.

# Glossary

Glossary of terms used in Adaptive Server ODBC Driver.

- **Adaptive Server® Enterprise –** a high-performance relational database management system for mission-critical, data-intensive environments. It ensures highest operational efficiency and throughput on a broad range of platforms.
- **American National Standards Institution (ANSI) code –** a standardized set of numeric or alphabetic codes issued by the ANSI to ensure uniform identification of geographic entities through all federal government agencies.
- **compiler –** a compiler is a program that translates a source program written in some high-level programming language into machine code.
- **connection string –** a string that specifies information about a data source and the means of connecting to it.
- **cursor –** a cursor is a data selector that passes multiple rows of data to the host program, one row at a time. The cursor indicates the first row, also called the current row of data and passes it to the host program.
- **data source name (DSN) –** a data structure that contains information about a specific database that an Open Database Connectivity (ODBC) driver needs to connect to.
- **descriptor –** a collection of metadata that describes the parameters of a SQL statement or the columns of a result set, as seen by the application or driver.
- **directory service URL (DSURL) –** a property called DSURL that indicates which LDAP server to use.
- **distributed transaction protocol –**
- **graphical user interface (GUI) –** a program interface that makes use of the computer's graphics capabilities to make the program easier to use.
- **ISO 8859-1 –** part of the ISO/IEC 8859 series of ASCII-based standard character encodings. It is informally referred to as Latin-1.
- **Kerberos authentication –** a mechanism for authentication and mutual authentication between a client and a server, or between one server and another server.
- **LDAP –** Lightweight Directory Access Protocol. An application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.
- **Mainframe Connect Direct Connect –** supplies the connectivity tools that allows full integration of mainframe and LAN-based data sources.
- **Microsoft distributed transaction coordinator (MS DTC) –** a component that coordinates transactions, which span multiple resource managers, such as databases, message queues, and file systems.
- **ODBC –** Open Database Connectivity. An open standard application programming interface (API) used for accessing a database.

- **ODBC Driver Manager –** the component that manages the communications between user applications and ODBC Drivers.
- **Powerbuilder –** a popular rapid application development (RAD) tool for building object-oriented client/server applications, the parts of which can be distributedwithin a network.
- **protocol data units (PDU) –** packages used for communication between clients and servers.
- **Secure Sockets Layer (SSL)/Transport Layer Security (TLS) –** SSL and TLS are cryptographic protocols that provide communication security over the Internet.
- **Sybase EA Server –** the leading solution for distributed and Web-enabled PowerBuilder application that leverages a new modular architecture to support custom deployments.
- **Sybase iAnywhere –** a software entity specializing in mobility (mobile computing), management, security and enterprise caliber database software.
- **Sybase Software Development Kit (SDK) –** the SDK provides a programming interface that allows transparent access to any data source, information application, or system services.
- **unicode –** a computing industry standard for the consistent encoding, representation and handling of text.

# Index

## F



## G



## H



## I



## K



## L



## M



## N

## O



## P



## R



## S

## T



## U



## V



## W