



# SQL Anywhere® Server Programming

**Version 12.0.1**

**January 2012**

Version 12.0.1  
January 2012

Copyright © 2012 iAnywhere Solutions, Inc. Portions copyright © 2012 Sybase, Inc. All rights reserved.

This documentation is provided AS IS, without warranty or liability of any kind (unless provided by a separate written agreement between you and iAnywhere).

You may use, print, reproduce, and distribute this documentation (in whole or in part) subject to the following conditions: 1) you must retain this and all other proprietary notices, on all copies of the documentation or portions thereof, 2) you may not modify the documentation, 3) you may not do anything to indicate that you or anyone other than iAnywhere is the author or source of the documentation.

iAnywhere®, Sybase®, and the marks listed at <http://www.sybase.com/detail?id=1011207> are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

---

---

# Contents

About this book .....	ix
Using SQL in applications .....	1
SQL statement execution in applications .....	1
Prepared statements .....	2
Cursor usage .....	5
Working with cursors .....	7
Cursor types .....	14
SQL Anywhere cursors .....	16
Result set descriptors .....	33
Transactions in applications .....	34
.NET application programming .....	39
SQL Anywhere .NET Data Provider .....	39
Using the SQL Anywhere .NET Data Provider .....	75
SQL Anywhere ASP.NET Providers .....	91
SQL Anywhere .NET API reference .....	99
OLE DB and ADO development .....	329
OLE DB .....	329
ADO programming with SQL Anywhere .....	330
OLE DB connection parameters .....	336
OLE DB connection pooling .....	338
Configuration of a Microsoft Linked Server using OLE DB .....	339
Supported OLE DB interfaces .....	342
Registering the OLE DB provider .....	346
ODBC support .....	347
Requirements for developing ODBC applications .....	347
Building ODBC applications .....	348

ODBC samples .....	354
ODBC handles .....	355
ODBC connection functions .....	357
Server options changed by ODBC .....	361
SQLSetConnectAttr extended connection attributes .....	361
Calling ODBC functions from DllMain .....	363
Ways to execute SQL statements .....	364
64-bit ODBC considerations .....	368
Data alignment requirements .....	372
Result sets in ODBC applications .....	374
Calling stored procedures .....	378
ODBC escape syntax .....	380
Error handling in ODBC .....	382
<b>Java in the database .....</b>	<b>385</b>
Learning about Java in the database .....	385
Java in the database FAQ .....	386
Java error handling .....	388
Creating a Java class for use with SQL Anywhere .....	388
Java VMs .....	389
Installing the sample Java class .....	390
The CLASSPATH environment variable .....	391
Calling methods in the Java class .....	391
Installing Java classes into a database .....	393
Special features of Java classes in the database .....	397
Starting and stopping the Java VM .....	401
Shutdown hooks in the Java VM .....	401
<b>JDBC support .....</b>	<b>403</b>
JDBC applications .....	403
JDBC drivers .....	404
JDBC program structure .....	406
Differences between client- and server-side JDBC connections .....	406
SQL Anywhere JDBC drivers .....	407

---

The jConnect JDBC driver .....	409
Connections from a JDBC client application .....	412
Data access using JDBC .....	419
JDBC callbacks .....	427
JDBC escape syntax .....	430
JDBC 3.0/4.0 API support .....	434
<b>Embedded SQL .....</b>	<b>435</b>
Development process overview .....	436
The SQL preprocessor .....	437
Supported compilers .....	441
Embedded SQL header files .....	441
Import libraries .....	442
Sample embedded SQL program .....	443
Structure of embedded SQL programs .....	443
DBLIB dynamic loading under Windows .....	444
Sample embedded SQL programs .....	445
Embedded SQL data types .....	449
Host variables in embedded SQL .....	452
The SQL Communication Area (SQLCA) .....	461
Static and dynamic SQL .....	466
The SQL descriptor area (SQLDA) .....	469
Fetching data .....	479
Sending and retrieving long values .....	487
Simple stored procedures in embedded SQL .....	491
Request management with embedded SQL .....	494
Database backup with embedded SQL .....	494
Library function reference .....	495
Embedded SQL statement summary .....	525
<b>SQL Anywhere database API for C/C++ .....</b>	<b>527</b>
SQL Anywhere C API support .....	527
SQL Anywhere C API reference .....	528

<b>SQL Anywhere external call interface .....</b>	<b>563</b>
Procedures and functions that use external calls .....	563
External function prototypes .....	565
External function call interface methods .....	574
Data type handling .....	578
Unloading external libraries .....	580
<b>SQL Anywhere external environment support .....</b>	<b>581</b>
The CLR external environment .....	584
The ESQL and ODBC external environments .....	587
The Java external environment .....	596
The PERL external environment .....	601
The PHP external environment .....	605
<b>Perl DBI support .....</b>	<b>611</b>
DBD::SQLAnywhere .....	611
Installing Perl/DBI support on Windows .....	611
Installing Perl/DBI support on Unix and Mac OS X .....	613
Perl scripts that use DBD::SQLAnywhere .....	614
<b>Python support .....</b>	<b>619</b>
sqlanydb .....	619
Installing Python support on Windows .....	620
Installing Python support on Unix and Mac OS X .....	620
Python scripts that use sqlanydb .....	621
<b>PHP support .....</b>	<b>627</b>
SQL Anywhere PHP extension .....	627
SQL Anywhere PHP API reference .....	640
<b>Ruby support .....</b>	<b>691</b>
SQL Anywhere Ruby API support .....	691

---

SQL Anywhere Ruby API reference .....	699
<b>Sybase Open Client support .....</b>	<b>721</b>
Open Client architecture .....	721
What you need to build Open Client applications .....	722
Data type mappings .....	723
SQL in Open Client applications .....	724
Known Open Client limitations of SQL Anywhere .....	727
<b>HTTP web services .....</b>	<b>729</b>
SQL Anywhere as an HTTP web server .....	729
Accessing web services using web clients .....	766
Web services references .....	803
HTTP web service examples .....	806
<b>Three-tier computing and distributed transactions .....</b>	<b>839</b>
Three-tier computing architecture .....	839
Distributed transactions .....	843
<b>Database tools interface (DBTools) .....</b>	<b>845</b>
DBTools import libraries .....	846
Initializing and finalizing the DBTools library .....	846
Calling the DBTools functions .....	847
Callback functions .....	847
Version numbers and compatibility .....	849
Bit fields .....	849
A DBTools example .....	850
SQL Anywhere Database Tools C API reference .....	852
Software component exit codes .....	912
<b>Database and application deployment .....</b>	<b>915</b>
Types of deployment .....	915

<b>Ways to distribute files .....</b>	<b>916</b>
<b>Installation directories and file names .....</b>	<b>916</b>
<b>The Deployment Wizard .....</b>	<b>919</b>
<b>Silent installs using the SQL Anywhere installer .....</b>	<b>923</b>
<b>Requirements for deploying client applications .....</b>	<b>927</b>
<b>Administration tool deployment .....</b>	<b>956</b>
<b>Documentation deployment .....</b>	<b>978</b>
<b>Database server deployment .....</b>	<b>979</b>
<b>Registering DLLs on Windows .....</b>	<b>984</b>
<b>External environment support deployment .....</b>	<b>985</b>
<b>Security deployment .....</b>	<b>987</b>
<b>Embedded database application deployment .....</b>	<b>988</b>
<b>Index .....</b>	<b>993</b>



---

# About this book

This book describes how to build and deploy database applications using the C, C++, Java, Perl, PHP, Python, Ruby, and .NET programming languages such as Visual Basic and Visual C#. A variety of programming interfaces such as ADO.NET, OLEDB, and ODBC are described.

---

---

# Using SQL in applications

This section provides information about using SQL in applications.

## SQL statement execution in applications

The way you include SQL statements in your application depends on the application development tool and programming interface you use.

- **ADO.NET** You can execute SQL statements using a variety of ADO.NET objects. The `SACommand` object is one example:

```
SACommand cmd = new SACommand(
    "DELETE FROM Employees WHERE EmployeeID = 105", conn );
cmd.ExecuteNonQuery();
```

- **ODBC** If you are writing directly to the ODBC programming interface, your SQL statements appear in function calls. For example, the following C function call executes a DELETE statement:

```
SQLExecDirect( stmt,
    "DELETE FROM Employees
    WHERE EmployeeID = 105",
    SQL_NTS );
```

- **JDBC** If you are using the JDBC programming interface, you can execute SQL statements by invoking methods of the statement object. For example:

```
stmt.executeUpdate(
    "DELETE FROM Employees
    WHERE EmployeeID = 105" );
```

- **Embedded SQL** If you are using embedded SQL, you prefix your C language SQL statements with the keyword `EXEC SQL`. The code is then run through a preprocessor before compiling. For example:

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees
WHERE EmployeeID = 105';
```

- **Sybase Open Client** If you use the Sybase Open Client interface, your SQL statements appear in function calls. For example, the following pair of calls executes a DELETE statement:

```
ret = ct_command( cmd, CS_LANG_CMD,
    "DELETE FROM Employees
    WHERE EmployeeID=105"
    CS_NULLTERM,
    CS_UNUSED);
ret = ct_send(cmd);
```

For more details about including SQL in your application, see your development tool documentation. If you are using ODBC or JDBC, consult the software development kit for those interfaces.

## Applications inside the database server

In many ways, stored procedures and triggers act as applications or parts of applications running inside the database server. You can also use many of the techniques here in stored procedures.

Java classes in the database can use the JDBC interface in the same way as Java applications outside the server. This section discusses some aspects of JDBC.

### See also

- [“SQL Anywhere .NET Data Provider” on page 39](#)
- [“ODBC support” on page 347](#)
- [“JDBC support” on page 403](#)
- [“Embedded SQL” on page 435](#)
- [“Sybase Open Client support” on page 721](#)
- [“Stored procedures, triggers, batches, and user defined functions” \[\*SQL Anywhere Server - SQL Usage\*\]](#)
- [“JDBC support” on page 403](#)

## Prepared statements

Each time a statement is sent to a database, the database server must perform the following steps:

- It must parse the statement and transform it into an internal form. This is sometimes called **preparing** the statement.
- It must verify the correctness of all references to database objects by checking, for example, that columns named in a query actually exist.
- The query optimizer generates an access plan if the statement involves joins or subqueries.
- It executes the statement after all these steps have been carried out.

### Reusing prepared statements can improve performance

If you use the same statement repeatedly, for example inserting many rows into a table, repeatedly preparing the statement causes a significant and unnecessary overhead. To remove this overhead, some database programming interfaces provide ways of using prepared statements. A **prepared statement** is a statement containing a series of placeholders. When you want to execute the statement, all you have to do is assign values to the placeholders, rather than prepare the entire statement over again.

Using prepared statements is particularly useful when carrying out many similar actions, such as inserting many rows.

Generally, using prepared statements requires the following steps:

1. **Prepare the statement** In this step you generally provide the statement with some placeholder character instead of the values.

2. **Repeatedly execute the prepared statement** In this step you supply values to be used each time the statement is executed. The statement does not have to be prepared each time.
3. **Drop the statement** In this step you free the resources associated with the prepared statement. Some programming interfaces handle this step automatically.

### **Do not prepare statements that are used only once**

In general, you should not prepare statements if they are only executed once. There is a slight performance penalty for separate preparation and execution, and it introduces unnecessary complexity into your application.

In some interfaces, however, you do need to prepare a statement to associate it with a cursor.

The calls for preparing and executing statements are not a part of SQL, and they differ from interface to interface. Each of the SQL Anywhere programming interfaces provides a method for using prepared statements.

### **See also**

- [“Cursor usage” on page 5](#)

## **How to use prepared statements**

This section provides a brief overview of how to use prepared statements. The general procedure is the same, but the details vary from interface to interface. Comparing how to use prepared statements in different interfaces illustrates this point.

### **Use a prepared statement (generic)**

1. Prepare the statement.
2. Bind the parameters that will hold values in the statement.
3. Assign values to the bound parameters in the statement.
4. Execute the statement.
5. Repeat steps 3 and 4 as needed.
6. Drop the statement when finished. In JDBC the Java garbage collection mechanism drops the statement.

### **Use a prepared statement (ADO.NET)**

1. Create an `SACommand` object holding the statement.

```
SACommand cmd = new SACommand(  
    "SELECT * FROM Employees WHERE Surname=?", conn );
```

2. Declare data types for any parameters in the statement.

Use the `SACCommand.CreateParameter` method.

3. Prepare the statement using the `Prepare` method.

```
cmd.Prepare();
```

4. Execute the statement.

```
SADataReader reader = cmd.ExecuteReader();
```

For an example of preparing statements using ADO.NET, see the source code in `%SQLANYAMP12%\SQLAnywhere\ADO.NET\SimpleWin32`.

### Use a prepared statement (ODBC)

1. Prepare the statement using `SQLPrepare`.
2. Bind the statement parameters using `SQLBindParameter`.
3. Execute the statement using `SQLExecute`.
4. Drop the statement using `SQLFreeStmt`.

For an example of preparing statements using ODBC, see the source code in `%SQLANYAMP12%\SQLAnywhere\ODBCPrepare`.

For more information about ODBC prepared statements, see the ODBC SDK documentation, and [“Executing prepared statements” on page 367](#).

### Use a prepared statement (JDBC)

1. Prepare the statement using the `prepareStatement` method of the connection object. This returns a prepared statement object.
2. Set the statement parameters using the appropriate `setType` methods of the prepared statement object. Here, *Type* is the data type assigned.
3. Execute the statement using the appropriate method of the prepared statement object. For inserts, updates, and deletes this is the `executeUpdate` method.

For an example of preparing statements using JDBC, see the source code file `%SQLANYAMP12%\SQLAnywhere\JDBC\JDBCExample.java`.

For more information about using prepared statements in JDBC, see [“Using prepared statements for more efficient access” on page 422](#).

### Use a prepared statement (embedded SQL)

1. Prepare the statement using the EXEC SQL PREPARE statement.

2. Assign values to the parameters in the statement.
3. Execute the statement using the EXEC SQL EXECUTE statement.
4. Free the resources associated with the statement using the EXEC SQL DROP statement.

For more information about embedded SQL prepared statements, see [“PREPARE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#).

### Use a prepared statement (Open Client)

1. Prepare the statement using the ct\_dynamic function, with a CS\_PREPARE type parameter.
2. Set statement parameters using ct\_param.
3. Execute the statement using ct\_dynamic with a CS\_EXECUTE type parameter.
4. Free the resources associated with the statement using ct\_dynamic with a CS\_DEALLOC type parameter.

For more information about using prepared statements in Open Client, see [“SQL in Open Client applications” on page 724](#).

## Cursor usage

When you execute a query in an application, the result set consists of several rows. In general, you do not know how many rows the application is going to receive before you execute the query. Cursors provide a way of handling query result sets in applications.

The way you use cursors and the kinds of cursors available to you depend on the programming interface you use. For a list of cursor types available from each interface, see [“Availability of cursors” on page 14](#).

SQL Anywhere provides several system procedures to help determine what cursors are in use for a connection, and what they contain:

- [“sa\\_list\\_cursors system procedure” \[SQL Anywhere Server - SQL Reference\]](#)
- [“sa\\_describe\\_cursor system procedure” \[SQL Anywhere Server - SQL Reference\]](#)
- [“sa\\_copy\\_cursor\\_to\\_temp\\_table system procedure” \[SQL Anywhere Server - SQL Reference\]](#)

With cursors, you can perform the following tasks within any programming interface:

- Loop over the results of a query.
- Perform inserts, updates, and deletes on the underlying data at any point within a result set.

In addition, some programming interfaces allow you to use special features to tune the way result sets return to your application, providing substantial performance benefits for your application.

For more information about the kinds of cursors available through different programming interfaces, see [“Availability of cursors” on page 14](#).

## Cursors

A **cursor** is a name associated with a result set. The result set is obtained from a `SELECT` statement or stored procedure call.

A cursor is a handle on the result set. At any time, the cursor has a well-defined position within the result set. With a cursor you can examine and possibly manipulate the data one row at a time. SQL Anywhere cursors support forward and backward movement through the query results.

### Cursor positions

Cursors can be positioned in the following places:

- Before the first row of the result set.
- On a row in the result set.
- After the last row of the result set.

Absolute row from start		Absolute row from end
0	Before first row	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		$-3$
$n - 1$		$-2$
$n$		$-1$
$n + 1$	After last row	0



The cursor position and result set are maintained in the database server. Rows are **fetched** by the client for display and processing either one at a time or a few at a time. The entire result set does not need to be delivered to the client.

## Benefits of using cursors

You do not need to use cursors in database applications, but they do provide several benefits. These benefits follow from the fact that if you do not use a cursor, the entire result set must be transferred to the client for processing and display:

- **Client-side memory** For large results, holding the entire result set on the client can lead to demanding memory requirements.
- **Response time** Cursors can provide the first few rows before the whole result set is assembled. If you do not use cursors, the entire result set must be delivered before any rows are displayed by your application.
- **Concurrency control** If you make updates to your data and do not use cursors in your application, you must send separate SQL statements to the database server to apply the changes. This raises the possibility of concurrency problems if the result set has changed since it was queried by the client. In turn, this raises the possibility of lost updates.

Cursors act as pointers to the underlying data, and so impose proper concurrency constraints on any changes you make.

## Working with cursors

This section describes how to perform different kinds of operations using cursors.

### Using cursors

Using a cursor in embedded SQL is different than using a cursor in other interfaces.

#### Use a cursor (embedded SQL)

1. Prepare a statement.

Cursors generally use a statement handle rather than a string. You need to prepare a statement to have a handle available.

For information about preparing a statement, see [“Prepared statements” on page 2](#).

2. Declare the cursor.

Each cursor refers to a single `SELECT` or `CALL` statement. When you declare a cursor, you state the name of the cursor and the statement it refers to.

For more information, see [“DECLARE CURSOR statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#).

3. Open the cursor. See [“OPEN statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#).

For a CALL statement, opening the cursor executes the procedure up to the point where the first row is about to be obtained.

4. Fetch results.

Although simple fetch operations move the cursor to the next row in the result set, SQL Anywhere permits more complicated movement around the result set. How you declare the cursor determines which fetch operations are available to you. See [“FETCH statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#) and [“Fetching data” on page 479](#).

5. Close the cursor.

When you have finished with the cursor, close it. This frees any resources associated with the cursor. See [“CLOSE statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#).

6. Drop the statement.

To free the memory associated with the statement, you must drop the statement. See [“DROP STATEMENT statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#).

For more information about using cursors in embedded SQL, see [“Fetching data” on page 479](#).

### **Use a cursor (ADO.NET, ODBC, JDBC, and Open Client)**

1. Prepare and execute a statement.

Execute a statement using the usual method for the interface. You can prepare and then execute the statement, or you can execute the statement directly.

With ADO.NET, only the `SACCommand.ExecuteReader` method returns a cursor. It provides a read-only, forward-only cursor.

2. Test to see if the statement returns a result set.

A cursor is implicitly opened when a statement that creates a result set is executed. When the cursor is opened, it is positioned before the first row of the result set.

3. Fetch results.

Although simple fetch operations move the cursor to the next row in the result set, SQL Anywhere permits more complicated movement around the result set.

4. Close the cursor.

When you have finished with the cursor, close it to free associated resources.

5. Free the statement.

If you used a prepared statement, free it to reclaim memory.

## Cursor positioning

When a cursor is opened, it is positioned before the first row. You can move the cursor position to an absolute position from the start or the end of the query results, or to a position relative to the current cursor position. The specifics of how you change cursor position, and what operations are possible, are governed by the programming interface.

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in an integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

You can use special positioned update and delete operations to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding cursor row.

### Note

Inserts and some updates to asensitive cursors can cause problems with cursor positioning. SQL Anywhere does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear at all until the cursor is closed and opened again. With SQL Anywhere, this occurs if a work table had to be created to open the cursor.

The UPDATE statement may cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a work table is not created). Using STATIC SCROLL cursors alleviates these problems but requires more memory and processing.

### See also

- “Use work tables in query processing (use All-rows optimization goal)” [[SQL Anywhere Server - SQL Usage](#)]

## Cursor behavior when opening cursors

You can configure the following aspects of cursor behavior when you open the cursor:

- **Isolation level** You can explicitly set the isolation level of operations on a cursor to be different from the current isolation level of the transaction. To do this, set the `isolation_level` option.
- **Holding** By default, cursors in embedded SQL close at the end of a transaction. Opening a cursor WITH HOLD allows you to keep it open until the end of a connection, or until you explicitly close it. ADO.NET, ODBC, JDBC, and Open Client leave cursors open at the end of transactions by default.

### See also

- “[isolation\\_level option](#)” [[SQL Anywhere Server - Database Administration](#)]

## Row fetching through a cursor

The simplest way of processing the result set of a query using a cursor is to loop through all the rows of the result set until there are no more rows.

### Loop through the rows of a result set

1. Declare and open the cursor (embedded SQL), or execute a statement that returns a result set (ODBC, JDBC, Open Client) or `SADaReader` object (ADO.NET).
2. Continue to fetch the next row until you get a `Row Not Found` error.
3. Close the cursor.

How step 2 of this operation is carried out depends on the interface you use. For example:

- **ADO.NET** Use the `SADaReader.NextResult` method. See “[SADaReader.NextResult method \[SQL Anywhere .NET\]](#)” on page 236.
- **ODBC** `SQLFetch`, `SQLExtendedFetch`, or `SQLFetchScroll` advances the cursor to the next row and returns the data.

For more information about using cursors in ODBC, see “[Result sets in ODBC applications](#)” on page 374.

- **JDBC** The next method of the `ResultSet` object advances the cursor and returns the data.

For more information about using the `ResultSet` object in JDBC, see “[Returning result sets](#)” on page 425.

- **Embedded SQL** The `FETCH` statement carries out the same operation.

For more information about using cursors in embedded SQL, see “[Cursors in embedded SQL](#)” on page 480.

- **Open Client** The `ct_fetch` function advances the cursor to the next row and returns the data.

For more information about using cursors in Open Client applications, see “[Cursors and the Open Client interface](#)” on page 725.

## Multiple-row fetching

Multiple-row fetching should not be confused with prefetching rows. Multiple row fetching is performed by the application, while prefetching is transparent to the application, and provides a similar performance gain. Fetching multiple rows at a time can improve performance.

### Multiple-row fetches

Some interfaces provide methods for fetching more than one row at a time into the next several fields in an array. Generally, the fewer separate fetch operations you execute, the fewer individual requests the server must respond to, and the better the performance. A modified FETCH statement that retrieves multiple rows is also sometimes called a **wide fetch**. Cursors that use multiple-row fetches are sometimes called **block cursors** or **fat cursors**.

### Using multiple-row fetching

- In ODBC, you can set the number of rows that will be returned on each call to SQLFetchScroll or SQLExtendedFetch by setting the SQL\_ATTR\_ROW\_ARRAY\_SIZE or SQL\_ROWSET\_SIZE attribute.
- In embedded SQL, the FETCH statement uses an ARRAY clause to control the number of rows fetched at a time.
- Open Client and JDBC do not support multi-row fetches. They do use prefetching.

## Scrollable cursors

ODBC and embedded SQL provide methods for using scrollable cursors and dynamic scrollable cursors. These methods allow you to move several rows forward at a time, or to move backward through the result set.

The JDBC and Open Client interfaces do not support scrollable cursors.

Prefetching does not apply to scrollable operations. For example, fetching a row in the reverse direction does not prefetch several previous rows.

## Cursors used to modify rows

Cursors can do more than just read result sets from a query. You can also modify data in the database while processing a cursor. These operations are commonly called **positioned** insert, update, and delete operations, or PUT operations if the action is an insert.

Not all query result sets allow positioned updates and deletes. If you perform a query on a non-updatable view, then no changes occur to the underlying tables. Also, if the query involves a join, then you must specify which table you want to delete from, or which columns you want to update, when you perform the operations.

Inserts through a cursor can only be executed if any non-inserted columns in the table allow NULL or have defaults.

If multiple rows are inserted into a value-sensitive (keyset driven) cursor, they appear at the end of the cursor result set. The rows appear at the end, even if they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set. This behavior is independent of programming interface. For example, it applies when using the embedded SQL PUT statement or the ODBC SQLBulkOperations function. The value of an AUTOINCREMENT column for the most recent row inserted can be found by selecting the last row in the cursor. For example, in embedded SQL the value could be obtained using `FETCH ABSOLUTE -1 cursor-name`. As a result of this behavior, the first multiple-row insert for a value-sensitive cursor may be expensive.

ODBC, JDBC, embedded SQL, and Open Client permit data manipulation using cursors, but ADO.NET does not. With Open Client, you can delete and update rows, but you can only insert rows on a single-table query.

### Which table are rows deleted from?

If you attempt a positioned delete through a cursor, the table from which rows are deleted is determined as follows:

1. If no FROM clause is included in the DELETE statement, the cursor must be on a single table only.
2. If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.
3. If a FROM clause is included, and no table owner is specified, the table-spec value is first matched against any correlation names.
4. If a correlation name exists, the table-spec value is identified with the correlation name.
5. If a correlation name does not exist, the table-spec value must be unambiguously identifiable as a table name in the cursor.
6. If a FROM clause is included, and a table owner is specified, the table-spec value must be unambiguously identifiable as a table name in the cursor.
7. The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

### See also

- [“FROM clause” \[SQL Anywhere Server - SQL Reference\]](#)

## Updatable statements

This section describes how clauses in the SELECT statement affect updatable statements and cursors.

### Updatability of read-only statements

Specifying FOR READ ONLY in the cursor declaration, or including a FOR READ ONLY clause in the statement, renders the statement read-only. In other words, a FOR READ ONLY clause, or the appropriate read-only cursor declaration when using a client API, overrides any other updatability specification.

If the outermost block of a SELECT statement contains an ORDER BY clause, and the statement does not specify FOR UPDATE, then the cursor is read-only. If the SQL SELECT statement specifies FOR XML, then the cursor is read-only. Otherwise, the cursor is updatable.

### Updatable statements and concurrency control

For updatable statements, SQL Anywhere provides both optimistic and pessimistic concurrency control mechanisms on cursors to ensure that a result set remains consistent during scrolling operations. These mechanisms are alternatives to using INSENSITIVE cursors or snapshot isolation, although they have different semantics and tradeoffs.

The specification of FOR UPDATE can affect whether a cursor is updatable. However, in SQL Anywhere, the FOR UPDATE syntax has no other effect on concurrency control. If FOR UPDATE is specified with additional parameters, SQL Anywhere alters the processing of the statement to incorporate one of two concurrency control options as follows:

- **Pessimistic** For all rows fetched in the cursor's result set, the database server acquires intent row locks to prevent the rows from being updated by any other transaction.
- **Optimistic** The cursor type used by the database server is changed to a keyset-driven cursor (insensitive row membership, value-sensitive) so that the application can be informed when a row in the result has been modified or deleted by this, or any other transaction.

Pessimistic or optimistic concurrency is specified at the cursor level either through options with DECLARE CURSOR or FOR statements, or through the concurrency setting API for a specific programming interface. If a statement is updatable and the cursor does not specify a concurrency control mechanism, the statement's specification is used. The syntax is as follows:

- **FOR UPDATE BY LOCK** The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until transaction COMMIT or ROLLBACK.
- **FOR UPDATE BY { VALUES | TIMESTAMP }** The database server utilizes a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

### Restricting updatable statements

FOR UPDATE ( *column-list* ) enforces the restriction that only named result set attributes can be modified in a subsequent UPDATE WHERE CURRENT OF statement.

### See also

- “DECLARE statement” [[SQL Anywhere Server - SQL Reference](#)]
- “FOR statement” [[SQL Anywhere Server - SQL Reference](#)]

## Canceling cursor operations

You can cancel a request through an interface function. From Interactive SQL, you can cancel a request by clicking the **Interrupt SQL Statement** button on the toolbar (or by clicking **SQL » Stop**).

If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. After canceling the request, you must locate the cursor by its absolute position, or close it.

## Cursor types

This section describes mappings between SQL Anywhere cursors and the options available to you from the programming interfaces supported by SQL Anywhere.

### See also

- [“SQL Anywhere cursors” on page 16](#)

## Availability of cursors

Not all interfaces provide support for all types of cursors.

- ADO.NET provides only forward-only, read-only cursors.
- ADO/OLE DB and ODBC support all types of cursors.
- Embedded SQL supports all types of cursors.
- For JDBC:
  - The SQL Anywhere JDBC driver supports the JDBC 3.0 and JDBC 4.0 specifications and permits the declaration of insensitive, sensitive, and forward-only asensitive cursors.
  - jConnect supports the declaration of insensitive, sensitive, and forward-only asensitive cursors in the same manner as the SQL Anywhere JDBC driver. However, the underlying implementation of jConnect only supports asensitive cursor semantics.
- Sybase Open Client supports only asensitive cursors. Also, a severe performance penalty results when using updatable, non-unique cursors.

### See also

- [“Result sets in ODBC applications” on page 374](#)
- [“Requests for SQL Anywhere cursors” on page 31](#)



## Cursor properties

You request a cursor type, either explicitly or implicitly, from the programming interface. Different interface libraries offer different choices of cursor types. For example, JDBC and ODBC specify different cursor types.

Each cursor type is defined by several characteristics:

- **Uniqueness** Declaring a cursor to be unique forces the query to return all the columns required to uniquely identify each row. Often this means returning all the columns in the primary key. Any columns required but not specified are added to the result set. The default cursor type is non-unique.
- **Updatability** A cursor declared as read-only cannot be used in a positioned update or delete operation. The default cursor type is updatable.
- **Scrollability** You can declare cursors to behave different ways as you move through the result set. Some cursors can fetch only the current row or the following row. Others can move backward and forward through the result set.
- **Sensitivity** Changes to the database may or may not be visible through a cursor.

These characteristics may have significant side effects on performance and on database server memory usage.

SQL Anywhere makes available cursors with a variety of mixes of these characteristics. When you request a cursor of a given type, SQL Anywhere tries to match those characteristics.

There are some occasions when not all characteristics can be supplied. For example, insensitive cursors in SQL Anywhere must be read-only. If your application requests an updatable insensitive cursor, a different cursor type (value-sensitive) is supplied instead.

## Bookmarks and cursors

ODBC provides **bookmarks**, or values, used to identify rows in a cursor. SQL Anywhere supports bookmarks for value-sensitive and insensitive cursors. For example, this means that the ODBC cursor types `SQL_CURSOR_STATIC` and `SQL_CURSOR_KEYSET_DRIVEN` support bookmarks while cursor types `SQL_CURSOR_DYNAMIC` and `SQL_CURSOR_FORWARD_ONLY` do not.

## Block cursors

ODBC provides a cursor type called a block cursor. When you use a `BLOCK` cursor, you can use `SQLFetchScroll` or `SQLExtendedFetch` to fetch a block of rows, rather than a single row. Block cursors behave identically to embedded SQL `ARRAY` fetches.

## SQL Anywhere cursors

Any cursor, once opened, has an associated result set. The cursor is kept open for a length of time. During that time, the result set associated with the cursor may be changed, either through the cursor itself or, subject to isolation level requirements, by other transactions. Some cursors permit changes to the underlying data to be visible, while others do not reflect these changes. A sensitivity to changes to the underlying data causes different cursor behavior, or **cursor sensitivity**.

SQL Anywhere provides cursors with a variety of sensitivity characteristics. This section describes what sensitivity is, and describes the sensitivity characteristics of cursors.

This section assumes that you have read [“Cursors” on page 6](#).

### Membership, order, and value changes

Changes to the underlying data can affect the result set of a cursor in the following ways:

- **Membership** The set of rows in the result set, as identified by their primary key values.
- **Order** The order of the rows in the result set.
- **Value** The values of the rows in the result set.

For example, consider the following simple table with employee information (EmployeeID is the primary key column):

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

A cursor on the following query returns all results from the table in primary key order:

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

The membership of the result set could be changed by adding a new row or deleting a row. The values could be changed by changing one of the names in the table. The order could be changed by changing the primary key value of one of the employees.

### Visible and invisible changes

Subject to isolation level requirements, the membership, order, and values of the result set of a cursor can be changed after the cursor is opened. Depending on the type of cursor in use, the result set as seen by the application may or may not change to reflect these changes.

Changes to the underlying data may be **visible** or **invisible** through the cursor. A visible change is a change that is reflected in the result set of the cursor. Changes to the underlying data that are not reflected in the result set seen by the cursor are invisible.

## Cursor sensitivity

SQL Anywhere cursors are classified by their sensitivity to changes in the underlying data. In other words, cursor sensitivity is defined by the changes that are visible.

- **Insensitive cursors** The result set is fixed when the cursor is opened. No changes to the underlying data are visible.
- **Sensitive cursors** The result set can change after the cursor is opened. All changes to the underlying data are visible.
- **Asensitive cursors** Changes may be reflected in the membership, order, or values of the result set seen through the cursor, or may not be reflected at all. See [“Asensitive cursors” on page 24](#).
- **Value-sensitive cursors** Changes to the order or values of the underlying data are visible. The membership of the result set is fixed when the cursor is opened. See [“Value-sensitive cursors” on page 24](#).

The differing requirements on cursors place different constraints on execution and therefore affect performance.

### See also

- [“Insensitive cursors” on page 21](#)
- [“Sensitive cursors” on page 22](#)
- [“Cursor sensitivity and performance” on page 26](#)

## Cursor sensitivity example: A deleted row

This example uses a simple query to illustrate how different cursors respond to a row in the result set being deleted.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

EmployeeID	Surname
102	Whitney

EmployeeID	Surname
105	Cobb
160	Breault
...	...

2. The application fetches the first row through the cursor (102).
3. The application fetches the next row through the cursor (105).
4. A separate transaction deletes employee 102 (Whitney) and commits the change.

The results of cursor actions in this situation depend on the cursor sensitivity:

- **Insensitive cursors** The DELETE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

- **Sensitive cursors** The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns ROW NOT FOUND. There is no previous row.
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 160.

- **Value-sensitive cursors** The membership of the result set is fixed, and so row 105 is still the second row of the result set. The DELETE is reflected in the values of the cursor, and creates an effective hole in the result set.

Action	Result
Fetch previous row	Returns <code>No current row of cursor</code> . There is a hole in the cursor where the first row used to be.
Fetch the first row (absolute fetch)	Returns <code>No current row of cursor</code> . There is a hole in the cursor where the first row used to be.
Fetch the second row (absolute fetch)	Returns row 105.

- Asensitive cursors** For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

The benefit of asensitive cursors is that for many applications, sensitivity is unimportant. In particular, if you are using a forward-only, read-only cursor, no underlying changes are seen. Also, if you are running at a high isolation level, underlying changes are disallowed.

## Cursor sensitivity example: An updated row

This example uses a simple query to illustrate how different cursor types respond to a row in the result set being updated in such a way that the order of the result set is changed.

Consider the following sequence of events:

1. An application opens a cursor on the following query against the sample database.

```
SELECT EmployeeID, Surname
FROM Employees;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. The application fetches the first row through the cursor (102).
3. The application fetches the next row through the cursor (105).
4. A separate transaction updates the employee ID of employee 102 (Whitney) to 165 and commits the change.

The results of the cursor actions in this situation depend on the cursor sensitivity:

- **Insensitive cursors** The UPDATE is not reflected in either the membership or values of the results as seen through the cursor:

Action	Result
Fetch previous row	Returns the original copy of the row (102).
Fetch the first row (absolute fetch)	Returns the original copy of the row (102).
Fetch the second row (absolute fetch)	Returns the unchanged row (105).

- **Sensitive cursors** The membership of the result set has changed so that row 105 is now the first row in the result set:

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row. The cursor is moved to the position before the first row.
Fetch the first row (absolute fetch)	Returns row 105.
Fetch the second row (absolute fetch)	Returns row 160.

In addition, a fetch on a sensitive cursor returns a `SQLE_ROW_UPDATED_WARNING` warning if the row has changed since the last reading. The warning is given only once. Subsequent fetches of the same row do not produce the warning.

Similarly, a positioned update or delete through the cursor on a row since it was last fetched returns the `SQLE_ROW_UPDATED_SINCE_READ` error. An application must fetch the row again for an update or delete on a sensitive cursor to work.

An update to any column causes the warning/error, even if the column is not referenced by the cursor. For example, a cursor on a query returning Surname would report the update even if only the Salary column was modified.

- **Value-sensitive cursors** The membership of the result set is fixed, and so row 105 is still the second row of the result set. The UPDATE is reflected in the values of the cursor, and creates an effective "hole" in the result set.

Action	Result
Fetch previous row	Returns SQLCODE 100. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the first row (absolute fetch)	Returns SQLCODE -197. The membership of the result set has changed so that 105 is now the first row: The cursor is positioned on the hole: it is before row 105.
Fetch the second row (absolute fetch)	Returns row 105.

- **Asensitive cursors** For changes, the membership and values of the result set are indeterminate. The response to a fetch of the previous row, the first row, or the second row depends on the particular optimization method for the query, whether that method involved the formation of a work table, and whether the row being fetched was prefetched from the client.

**Note**

Update warning and error conditions do not occur in bulk operations mode (-b database server option).

**See also**

- “Row not found” [*Error Messages*]
- “Row has been updated since last time read” [*Error Messages*]
- “Row has changed since last read -- operation canceled” [*Error Messages*]
- “No current row of cursor” [*Error Messages*]

## Inensitive cursors

These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

Inensitive cursors are used only for read-only cursor types.

**Standards**

Inensitive cursors correspond to the ISO/ANSI standard definition of insensitive cursors, and to ODBC static cursors.

**Programming interfaces**

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Static	If an updatable static cursor is requested, a value-sensitive cursor is used instead.

Interface	Cursor type	Comment
Embedded SQL	INSENSITIVE	
JDBC	INSENSITIVE	Insensitive semantics are only supported by the SQL Anywhere JDBC driver.
Open Client	Unsupported	

### Description

Insensitive cursors always return rows that match the query's selection criteria, in the order specified by any ORDER BY clause.

The result set of an insensitive cursor is fully materialized as a work table when the cursor is opened. This has the following consequences:

- If the result set is very large, the disk space and memory requirements for managing the result set may be significant.
- No row is returned to the application before the entire result set is assembled as a work table. For complex queries, this may lead to a delay before the first row is returned to the application.
- Subsequent rows can be fetched directly from the work table, and so are returned quickly. The client library may prefetch several rows at a time, further improving performance.
- Insensitive cursors are not affected by ROLLBACK or ROLLBACK TO SAVEPOINT.

## Sensitive cursors

Sensitive cursors can be used for read-only or updatable cursor types.

These cursors have sensitive membership, order, and values.

### Standards

Sensitive cursors correspond to the ISO/ANSI standard definition of sensitive cursors, and to ODBC dynamic cursors.

### Programming interfaces

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Dynamic	
Embedded SQL	SENSITIVE	Also supplied in response to a request for a DYNAMIC SCROLL cursor when no work table is required and the prefetch option is set to Off.



Interface	Cursor type	Comment
JDBC	SENSITIVE	Sensitive cursors are fully supported by the SQL Anywhere JDBC driver.

**Description**

Prefetching is disabled for sensitive cursors. All changes are visible through the cursor, including changes through the cursor and from other transactions. Higher isolation levels may hide some changes made in other transactions because of locking.

Changes to cursor membership, order, and all column values are all visible. For example, if a sensitive cursor contains a join, and one of the values of one of the underlying tables is modified, then all result rows composed from that base row show the new value. Result set membership and order may change at each fetch.

Sensitive cursors always return rows that match the query's selection criteria, and are in the order specified by any ORDER BY clause. Updates may affect the membership, order, and values of the result set.

The requirements of sensitive cursors place restrictions on the implementation of sensitive cursors:

- Rows cannot be prefetched, as changes to the prefetched rows would not be visible through the cursor. This may impact performance.
- Sensitive cursors must be implemented without any work tables being constructed, as changes to those rows stored as work tables would not be visible through the cursor.
- The no work table limitation restricts the choice of join method by the optimizer and therefore may impact performance.
- For some queries, the optimizer is unable to construct a plan that does not include a work table that would make a cursor sensitive.

Work tables are commonly used for sorting and grouping intermediate results. A work table is not needed for sorting if the rows can be accessed through an index. It is not possible to state exactly which queries employ work tables, but the following queries do employ them:

- UNION queries, although UNION ALL queries do not necessarily use work tables.
- Statements with an ORDER BY clause, if there is no index on the ORDER BY column.
- Any query that is optimized using a hash join.
- Many queries involving DISTINCT or GROUP BY clauses.

In these cases, SQL Anywhere either returns an error to the application, or changes the cursor type to an asensitive cursor and returns a warning.

**See also**

- [“Use work tables in query processing \(use All-rows optimization goal\)” \[SQL Anywhere Server - SQL Usage\]](#)

## Asensitive cursors

These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

Asensitive cursors are used only for read-only cursor types.

**Standards**

Asensitive cursors correspond to the ISO/ANSI standard definition of asensitive cursors, and to ODBC cursors with unspecified sensitivity.

**Programming interfaces**

Interface	Cursor type
ODBC, ADO/OLE DB	Unspecified sensitivity
Embedded SQL	DYNAMIC SCROLL

**Description**

A request for an asensitive cursor places few restrictions on the methods SQL Anywhere can use to optimize the query and return rows to the application. For these reasons, asensitive cursors provide the best performance. In particular, the optimizer is free to employ any measure of materialization of intermediate results as work tables, and rows can be prefetched by the client.

SQL Anywhere makes no guarantees about the visibility of changes to base underlying rows. Some changes may be visible, others not. Membership and order may change at each fetch. In particular, updates to base rows may result in only some of the updated columns being reflected in the cursor's result.

Asensitive cursors do not guarantee to return rows that match the query's selection and order. The row membership is fixed at cursor open time, but subsequent changes to the underlying values are reflected in the results.

Asensitive cursors always return rows that matched the customer's WHERE and ORDER BY clauses at the time the cursor membership is established. If column values change after the cursor is opened, rows may be returned that no longer match WHERE and ORDER BY clauses.

## Value-sensitive cursors

For value-sensitive cursors, membership is insensitive, and the order and value of the result set is sensitive.

Value-sensitive cursors can be used for read-only or updatable cursor types.

## Standards

Value-sensitive cursors do not correspond to an ISO/ANSI standard definition. They correspond to ODBC keyset-driven cursors.

## Programming interfaces

Interface	Cursor type	Comment
ODBC, ADO/OLE DB	Keyset-driven	
Embedded SQL	SCROLL	
JDBC	INSENSITIVE and CONCUR_UPDATABLE	With the SQL Anywhere JDBC driver, a request for an updatable INSENSITIVE cursor is answered with a value-sensitive cursor.
Open Client and jConnect	Not supported	

## Description

If the application fetches a row composed of a base underlying row that has changed, then the application must be presented with the updated value, and the `SQL_ROW_UPDATED` status must be issued to the application. If the application attempts to fetch a row that was composed of a base underlying row that was deleted, a `SQL_ROW_DELETED` status must be issued to the application.

Changes to primary key values remove the row from the result set (treated as a delete, followed by an insert). A special case occurs when a row in the result set is deleted (either from cursor or outside) and a new row with the same key value is inserted. This will result in the new row replacing the old row where it appeared.

There is no guarantee that rows in the result set match the query's selection or order specification. Since row membership is fixed at open time, subsequent changes that make a row not match the `WHERE` clause or `ORDER BY` do not change a row's membership nor position.

All values are sensitive to changes made through the cursor. The sensitivity of membership to changes made through the cursor is controlled by the ODBC option `SQL_STATIC_SENSITIVITY`. If this option is on, then inserts through the cursor add the row to the cursor. Otherwise, they are not part of the result set. Deletes through the cursor remove the row from the result set, preventing a hole returning the `SQL_ROW_DELETED` status.

Value-sensitive cursors use a **key set table**. When the cursor is opened, SQL Anywhere populates a work table with identifying information for each row contributing to the result set. When scrolling through the result set, the key set table is used to identify the membership of the result set, but values are obtained, if necessary, from the underlying tables.

The fixed membership property of value-sensitive cursors allows your application to remember row positions within a cursor and be assured that these positions will not change. See [“Cursor sensitivity example: A deleted row” on page 17](#).

- If a row was updated or may have been updated since the cursor was opened, SQL Anywhere returns a `SQL_ROW_UPDATED_WARNING` when the row is fetched. The warning is generated only once: fetching the same row again does not produce the warning.

An update to any column of the row causes the warning, even if the updated column is not referenced by the cursor. For example, a cursor on Surname and GivenName would report the update even if only the Birthdate column was modified. These update warning and error conditions do not occur in bulk operations mode (-b database server option) when row locking is disabled. See [“Performance aspects of bulk operations” \[SQL Anywhere Server - SQL Usage\]](#), and [“Row has been updated since last time read” \[Error Messages\]](#).

- An attempt to execute a positioned update or delete on a row that has been modified since it was last fetched returns a `SQL_ROW_UPDATED_SINCE_READ` error and cancels the statement. An application must `FETCH` the row again before the `UPDATE` or `DELETE` is permitted.

An update to any column of the row causes the error, even if the updated column is not referenced by the cursor. The error does not occur in bulk operations mode. See [“Row has changed since last read -- operation canceled” \[Error Messages\]](#).

- If a row has been deleted after the cursor is opened, either through the cursor or from another transaction, a **hole** is created in the cursor. The membership of the cursor is fixed, so a row position is reserved, but the `DELETE` operation is reflected in the changed value of the row. If you fetch the row at this hole, you receive a -197 `SQLCODE` error, indicating that there is no current row, and the cursor is left positioned on the hole. You can avoid holes by using sensitive cursors, as their membership changes along with the values. See [“No current row of cursor” \[Error Messages\]](#).

Rows cannot be prefetched for value-sensitive cursors. This requirement may affect performance.

### Inserting multiple rows

When inserting multiple rows through a value-sensitive cursor, the new rows appear at the end of the result set. See [“Cursors used to modify rows” on page 11](#).

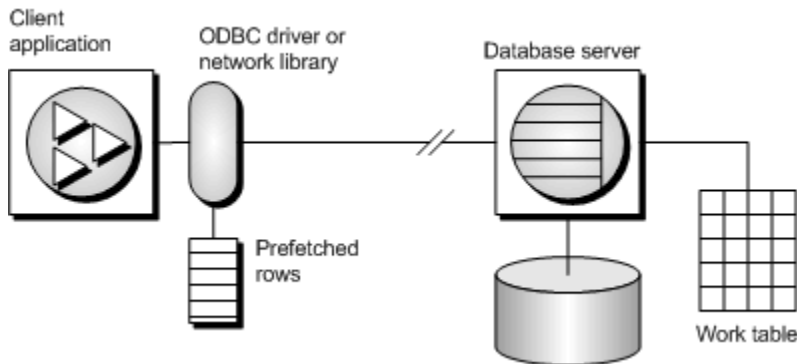
## Cursor sensitivity and performance

There is a trade-off between performance and other cursor properties. In particular, making a cursor updatable places restrictions on the cursor query processing and delivery that constrain performance. Also, putting requirements on cursor sensitivity may constrain cursor performance.

To understand how the updatability and sensitivity of cursors affects performance, you need to understand how the results that are visible through a cursor are transmitted from the database to the client application.

In particular, results may be stored at two intermediate locations for performance reasons:

- **Work tables** Either intermediate or final results may be stored as work tables. Value-sensitive cursors employ a work table of primary key values. Query characteristics may also lead the optimizer to use work tables in its chosen execution plan.
- **Prefetching** The client side of the communication may retrieve rows into a buffer on the client side to avoid separate requests to the database server for each row.



Sensitivity and updatability limit the use of intermediate locations.

## Prefetches

Prefetches and multiple-row fetches are different. Prefetches can be carried out without explicit instructions from the client application. Prefetching retrieves rows from the server into a buffer on the client side, but does not make those rows available to the client application until the application fetches the appropriate row.

By default, the SQL Anywhere client library prefetches multiple rows whenever an application fetches a single row. The SQL Anywhere client library stores the additional rows in a buffer.

Prefetching assists performance by cutting down on client/server round trips, and increases throughput by making many rows available without a separate request to the server for each row or block of rows.

For more information about controlling prefetches, see [“prefetch option” \[SQL Anywhere Server - Database Administration\]](#).

### Controlling prefetching from an application

- The prefetch option controls whether prefetching occurs. You can set the prefetch option to Always, Conditional, or Off for a single connection. By default, it is set to Conditional.
- In embedded SQL, you can control prefetching on a per-cursor basis when you open a cursor on an individual FETCH operation using the BLOCK clause.

The application can specify a maximum number of rows contained in a single fetch from the server by specifying the BLOCK clause. For example, if you are fetching and displaying 5 rows at a time, you could use BLOCK 5. Specifying BLOCK 0 fetches 1 record at a time and also causes a FETCH RELATIVE 0 to always fetch the row from the server again.

Although you can also turn off prefetch by setting a connection parameter on the application, it is more efficient to specify BLOCK 0 than to set the prefetch option to Off. See “[prefetch option](#)” [*SQL Anywhere Server - Database Administration*].

- Prefetch is disabled by default for value sensitive cursor types.
- In Open Client, you can control prefetching behavior using `ct_cursor` with `CS_CURSOR_ROWS` after the cursor is declared, but before it is opened.

Prefetch dynamically increases the number of prefetch rows when improvements in performance could be achieved. This includes cursors that meet the following conditions:

- They use one of the supported cursor types:
  - **ODBC and OLE DB** FORWARD-ONLY and READ-ONLY (default) cursors
  - **Embedded SQL** DYNAMIC SCROLL (default), NO SCROLL, and INSENSITIVE cursors
  - **ADO.NET** all cursors
- They perform only FETCH NEXT operations (no absolute, relative, or backward fetching).
- The application does not change the host variable type between fetches and does not use a GET DATA statement to get column data in chunks (using *one* GET DATA statement to get the value is supported).

## Lost updates

When using an updatable cursor, it is important to guard against lost updates. A lost update is a scenario in which two or more transactions update the same row, but neither transaction is aware of the modification made by the other transaction, and the second change overwrites the first modification. The following example illustrates this problem:

1. An application opens a cursor on the following query against the sample database.

```
SELECT ID, Quantity
FROM Products;
```

ID	Quantity
300	28
301	54
302	75
...	...

2. The application fetches the row with ID = 300 through the cursor.
3. A separate transaction updates the row using the following statement:

```
UPDATE Products
SET Quantity = Quantity - 10
WHERE ID = 300;
```

4. The application then updates the row through the cursor to a value of (Quantity - 5).
5. The correct final value for the row would be 13. If the cursor had prefetched the row, the new value of the row would be 23. The update from the separate transaction is lost.

In a database application, the potential for a lost update exists at any isolation level if changes are made to rows without verification of their values beforehand. At higher isolation levels (2 and 3), locking (read, intent, and write locks) can be used to ensure that changes to rows cannot be made by another transaction once the row has been read by the application. However, at isolation levels 0 and 1, the potential for lost updates is greater: at isolation level 0, read locks are not acquired to prevent subsequent changes to the data, and isolation level 1 only locks the current row. Lost updates cannot occur when using snapshot isolation since any attempt to change an old value results in an update conflict. Also, the use of prefetching at isolation level 1 can also introduce the potential for lost updates, since the result set row that the application is positioned on, which is in the client's prefetch buffer, may not be the same as the current row that the server is positioned on in the cursor.

To prevent lost updates from occurring with cursors at isolation level 1, the database server supports three different concurrency control mechanisms that can be specified by an application:

1. The acquisition of intent row locks on each row in the cursor as it is fetched. Intent locks prevent other transactions from acquiring intent or write locks on the same row, preventing simultaneous updates. However, intent locks do not block read row locks, so they do not affect the concurrency of read-only statements.
2. The use of a value-sensitive cursor. Value-sensitive cursors can be used to track when an underlying row has changed, or has been deleted, so that the application can respond.
3. The use of FETCH FOR UPDATE, which acquires an intent row lock for that specific row.

How these alternatives are specified depends on the interface used by the application. For the first two alternatives that pertain to a SELECT statement:

- In ODBC, lost updates cannot occur because the application must specify a cursor concurrency parameter to the SQLSetStmtAttr function when declaring an updatable cursor. This parameter is one of SQL\_CONCUR\_LOCK, SQL\_CONCUR\_VALUES, SQL\_CONCUR\_READ\_ONLY, or SQL\_CONCUR\_TIMESTAMP. For SQL\_CONCUR\_LOCK, the database server acquires row intent locks. For SQL\_CONCUR\_VALUES and SQL\_CONCUR\_TIMESTAMP, a value-sensitive cursor is used. SQL\_CONCUR\_READ\_ONLY is used for read-only cursors, and is the default.
- In JDBC, the concurrency setting for a statement is similar to that of ODBC. The SQL Anywhere JDBC driver supports the JDBC concurrency values RESULTSET\_CONCUR\_READ\_ONLY and RESULTSET\_CONCUR\_UPDATABLE. The first value corresponds to the ODBC concurrency setting SQL\_CONCUR\_READ\_ONLY and specifies a read-only statement. The second value corresponds to the ODBC SQL\_CONCUR\_LOCK setting, so row intent locks are used to prevent lost updates. Note that value-sensitive cursors cannot be specified directly in the JDBC 3.0 or 4.0 specification.

- In jConnect, updatable cursors are supported at the API level, but the underlying implementation (using TDS) does not support updates through a cursor. Instead, jConnect sends a separate UPDATE statement to the database server to update the specific row. To avoid lost updates, the application must run at isolation level 2 or higher. Alternatively, the application can issue separate UPDATE statements from the cursor, but you must ensure that the UPDATE statement verifies that the row values have not been altered since the row was read by placing appropriate conditions in the UPDATE statement's WHERE clause.

- In embedded SQL, a concurrency specification can be set by including syntax within the SELECT statement itself, or in the cursor declaration. In the SELECT statement, the syntax SELECT...FOR UPDATE BY LOCK causes the database server to acquire intent row locks on the result set.

Alternatively, SELECT...FOR UPDATE BY [ VALUES | TIMESTAMP ] causes the database server to change the cursor type to a value-sensitive cursor, so that if a specific row has been changed since the row was last read through the cursor, the application receives either a warning (SQLE\_ROW\_UPDATED\_WARNING) on a FETCH statement, or an error (SQLE\_ROW\_UPDATED\_SINCE\_READ) on an UPDATE WHERE CURRENT OF statement. If the row was deleted, the application also receives an error (SQLE\_NO\_CURRENT\_ROW).

FETCH FOR UPDATE functionality is also supported by the embedded SQL and ODBC interfaces, although the details differ depending on the API that is used.

In embedded SQL, the application uses FETCH FOR UPDATE, rather than FETCH, to cause an intent lock to be acquired on the row. In ODBC, the application uses the API call SQLSetPos with the operation argument SQL\_POSITION or SQL\_REFRESH, and the lock type argument SQL\_LOCK\_EXCLUSIVE, to acquire an intent lock on a row. In SQL Anywhere, these are long-term locks that are held until the transaction commits or rolls back.

## Cursor sensitivity and isolation levels

Both cursor sensitivity and isolation levels address the problem of concurrency control, but in different ways, and with different sets of tradeoffs.

By choosing an isolation level for a transaction (typically at the connection level), you determine the type and locks to place, and when, on rows in the database. Locks prevent other transactions from accessing or modifying rows in the database. In general, the greater the number of locks held, the lower the expected level of concurrency across concurrent transactions.

However, locks do not prevent updates from other portions of the same transaction from occurring. So, a single transaction that maintains multiple updatable cursors cannot rely on locking to prevent such problems as lost updates.

Snapshot isolation is intended to eliminate the need for read locks by ensuring that each transaction sees a consistent view of the database. The obvious advantage is that a consistent view of the database can be queried without relying on fully serializable transactions (isolation level 3), and the loss of concurrency that comes with using isolation level 3. However, snapshot isolation comes with a significant cost because copies of modified rows must be maintained to satisfy the requirements of both concurrent snapshot



transactions already executing, and snapshot transactions that have yet to start. Because of this copy maintenance, the use of snapshot isolation may be inappropriate for heavy-update workloads. See [“Choosing a snapshot isolation level” \[SQL Anywhere Server - SQL Usage\]](#).

Cursor sensitivity, however, determines which changes are visible (or not) to the cursor's result. Because cursor sensitivity is specified on a cursor basis, cursor sensitivity applies to both the effects of other transactions and to update activity of the same transaction, although these effects depend entirely on the cursor type specified. By setting cursor sensitivity, you are not directly determining when locks are placed on rows in the database. However, it is the combination of cursor sensitivity and isolation level that controls the various concurrency scenarios that are possible with a particular application.

## Requests for SQL Anywhere cursors

When you request a cursor type from your client application, SQL Anywhere provides a cursor. SQL Anywhere cursors are defined, not by the type as specified in the programming interface, but by the sensitivity of the result set to changes in the underlying data. Depending on the cursor type you ask for, SQL Anywhere provides a cursor with behavior to match the type.

SQL Anywhere cursor sensitivity is set in response to the client cursor type request.

## ADO.NET

Forward-only, read-only cursors are available by using `SACommand.ExecuteReader`. The `SADDataAdapter` object uses a client-side result set instead of cursors. See [“SACommand class \[SQL Anywhere .NET\]” on page 122](#).

## ADO/OLE DB and ODBC

The following table illustrates the cursor sensitivity that is set in response to different ODBC scrollable cursor types.

ODBC scrollable cursor type	SQL Anywhere cursor
STATIC	Insensitive
KEYSET-DRIVEN	Value-sensitive
DYNAMIC	Sensitive
MIXED	Value-sensitive

A MIXED cursor is obtained by setting the cursor type to `SQL_CURSOR_KEYSET_DRIVEN`, and then specifying the number of rows in the keyset for a keyset-driven cursor using `SQL_ATTR_KEYSET_SIZE`. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the

keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside the keyset). The default keyset size is 0. It is an error if the keyset size is greater than 0 and less than the rowset size (SQL\_ATTR\_ROW\_ARRAY\_SIZE).

For information about SQL Anywhere cursors and their behavior, see [“SQL Anywhere cursors” on page 16](#).

For information about how to request a cursor type in ODBC, see [“ODBC cursor characteristics” on page 375](#).

## Exceptions

If a STATIC cursor is requested as updatable, a value-sensitive cursor is supplied instead and a warning is issued.

If a DYNAMIC or MIXED cursor is requested and the query cannot be executed without using work tables, a warning is issued and an asensitive cursor is supplied instead.

## JDBC

The JDBC 3.0 and 4.0 specifications support three types of cursors: insensitive, sensitive, and forward-only asensitive. The SQL Anywhere JDBC driver is compliant with these JDBC specifications and supports these different cursor types for a JDBC ResultSet object. However, there are cases when the database server cannot construct an access plan with the required semantics for a given cursor type. In these cases, the database server either returns an error or substitutes a different cursor type. See [“Sensitive cursors” on page 22](#).

With jConnect, the underlying protocol (TDS) only supports forward-only, read-only asensitive cursors on the database server, even though jConnect supports the APIs for creating different types of cursors following the JDBC 2.0 specification. All jConnect cursors are asensitive because the TDS protocol buffers the statement's result set in blocks. These blocks of buffered results are scrolled when the application needs to scroll through an insensitive or sensitive cursor type that supports scrollability. If the application scrolls backward past the beginning of the cached result set, the statement is re-executed, which can result in data inconsistencies if the data has been altered between statement executions.

## Embedded SQL

To request a cursor from an embedded SQL application, you specify the cursor type on the DECLARE statement. The following table illustrates the cursor sensitivity that is set in response to different requests:

Cursor type	SQL Anywhere cursor
NO SCROLL	Asensitive
DYNAMIC SCROLL	Asensitive
SCROLL	Value-sensitive

Cursor type	SQL Anywhere cursor
INSENSITIVE	Insensitive
SENSITIVE	Sensitive

### Exceptions

If a DYNAMIC SCROLL or NO SCROLL cursor is requested as UPDATABLE, then a sensitive or value-sensitive cursor is supplied. It is not guaranteed which of the two is supplied. This uncertainty fits the definition of asensitive behavior.

If an INSENSITIVE cursor is requested as UPDATABLE, then a value-sensitive cursor is supplied.

If a DYNAMIC SCROLL cursor is requested, if the prefetch database option is set to Off, and if the query execution plan involves no work tables, then a sensitive cursor may be supplied. Again, this uncertainty fits the definition of asensitive behavior.

## Open Client

As with jConnect, the underlying protocol (TDS) for Open Client only supports forward-only, read-only, asensitive cursors.

## Result set descriptors

Some applications build SQL statements that cannot be completely specified in the application. Sometimes statements are dependent on a user response before the application knows exactly what information to retrieve, such as when a reporting application allows a user to select which columns to display.

In such a case, the application needs a method for retrieving information about both the nature of the **result set** and the contents of the result set. The information about the nature of the result set, called a **descriptor**, identifies the data structure, including the number and type of columns expected to be returned. Once the application has determined the nature of the result set, retrieving the contents is straightforward.

This **result set metadata** (information about the nature and content of the data) is manipulated using descriptors. Obtaining and managing the result set metadata is called **describing**.

Since cursors generally produce result sets, descriptors and cursors are closely linked, although some interfaces hide the use of descriptors from the user. Typically, statements needing descriptors are either SELECT statements or stored procedures that return result sets.

A sequence for using a descriptor with a cursor-based operation is as follows:

1. Allocate the descriptor. This may be done implicitly, although some interfaces allow explicit allocation as well.

2. Prepare the statement.
3. Describe the statement. If the statement is a stored procedure call or batch, and the result set is not defined by a result clause in the procedure definition, then the describe should occur after opening the cursor.
4. Declare and open a cursor for the statement (embedded SQL) or execute the statement.
5. Get the descriptor and modify the allocated area if necessary. This is often done implicitly.
6. Fetch and process the statement results.
7. Deallocate the descriptor.
8. Close the cursor.
9. Drop the statement. Some interfaces do this automatically.

### Implementation notes

- In embedded SQL, a SQLDA (SQL Descriptor Area) structure holds the descriptor information. See [“The SQL descriptor area \(SQLDA\)” on page 469](#).
- In ODBC, a descriptor handle allocated using `SQLAllocHandle` provides access to the fields of a descriptor. You can manipulate these fields using `SQLSetDescRec`, `SQLSetDescField`, `SQLGetDescRec`, and `SQLGetDescField`.  
Alternatively, you can use `SQLDescribeCol` and `SQLColAttributes` to obtain column information.
- In Open Client, you can use `ct_dynamic` to prepare a statement and `ct_describe` to describe the result set of the statement. However, you can also use `ct_command` to send a SQL statement without preparing it first and use `ct_results` to handle the returned rows one by one. This is the more common way of operating in Open Client application development.
- In JDBC, the `java.sql.ResultSetMetaData` class provides information about result sets.
- You can also use descriptors for sending data to the database server (for example, with the `INSERT` statement); however, this is a different kind of descriptor than for result sets.  
For more information about input and output parameters of the `DESCRIBE` statement, see [“DESCRIBE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#).

## Transactions in applications

Transactions are sets of atomic SQL statements. Either all statements in the transaction are executed, or none. This section describes a few aspects of transactions in applications.

For more information about transactions, see [“Transactions and isolation levels” \[SQL Anywhere Server - SQL Usage\]](#).

## Autocommit and manual commit mode

Database programming interfaces can operate in either **manual commit** mode or **autocommit** mode.

- **Manual commit mode** Operations are committed only when your application carries out an explicit commit operation or when the database server carries out an automatic commit, for example when executing an ALTER TABLE statement or other data definition statement. Manual commit mode is also sometimes called **chained mode**.

To use transactions in your application, including nested transactions and savepoints, you must operate in manual commit mode.

- **Autocommit mode** Each statement is treated as a separate transaction. Autocommit mode is equivalent to appending a COMMIT statement to the end of each of your SQL statements. Autocommit mode is also sometimes called **unchained mode**.

Autocommit mode can affect the performance and behavior of your application. Do not use autocommit if your application requires transactional integrity.

For information about how autocommit mode affects performance, see [“Turn off autocommit mode” \[SQL Anywhere Server - SQL Usage\]](#).

## Controlling autocommit behavior

The way to control the commit behavior of your application depends on the programming interface you are using. The implementation of autocommit may be client-side or server-side, depending on the interface. See [“Autocommit implementation details” on page 36](#).

### Control autocommit mode (ADO.NET)

- By default, the ADO.NET provider operates in autocommit mode. To use explicit transactions, use the `SACConnection.BeginTransaction` method. See [“Transaction processing” on page 62](#).

### Control autocommit mode (OLE DB)

- By default, the OLE DB provider operates in autocommit mode. To use explicit transactions, use the `ITransactionLocal::StartTransaction`, `ITransaction::Commit`, and `ITransaction::Abort` methods.

### Control autocommit mode (ODBC)

- By default, ODBC operates in autocommit mode. The way you turn off autocommit depends on whether you are using ODBC directly, or using an application development tool. If you are programming directly to the ODBC interface, set the `SQL_ATTR_AUTOCOMMIT` connection attribute.

### Control autocommit mode (JDBC)

- By default, JDBC operates in autocommit mode. To turn off autocommit, use the `setAutoCommit` method of the connection object:

```
conn.setAutoCommit( false );
```

### Control autocommit mode (embedded SQL)

- By default, embedded SQL applications operate in manual commit mode. To turn on autocommit, set the chained database option (a server-side option) to Off using a statement such as the following:

```
SET OPTION chained='Off';
```

### Control autocommit mode (Open Client)

- By default, a connection made through Open Client operates in autocommit mode. You can change this behavior by setting the chained database option (a server-side option) to On in your application using a statement such as the following:

```
SET OPTION chained='On';
```

### Control autocommit mode (PHP)

- By default, PHP operates in autocommit mode. To turn off autocommit, use the `sqlanywhere_set_option` function:

```
$result = sasql_set_option( $conn, "auto_commit", "Off" );
```

### Control autocommit mode (on the server)

- By default, the database server operates in manual commit mode. To turn on automatic commits, set the chained database option (a server-side option) to Off using a statement such as the following:

```
SET OPTION chained='Off';
```

If you are using an interface that controls commits on the client side, setting the chained database option (a server-side option) can impact performance and/or behavior of your application. Setting the server's chained mode is not recommended.

See [“Autocommit and manual commit mode”](#) on page 35.

## Autocommit implementation details

Autocommit mode has slightly different behavior depending on the interface and provider that you are using and how you control the autocommit behavior.

Autocommit mode can be implemented in one of two ways:

- **Client-side autocommit** When an application uses autocommit, the client-library sends a COMMIT statement after each SQL statement executed.

ADO.NET, ADO/OLE DB, ODBC, PHP, and SQL Anywhere JDBC driver applications control commit behavior from the client side.

- **Server-side autocommit** When an application turns off chained mode, the database server commits the results of each SQL statement. For the Sybase jConnect JDBC driver, this behavior is controlled by the chained database option.

Embedded SQL, the jConnect driver, and Open Client applications manipulate server-side commit behavior (for example, they set the chained option).

For compound statements such as stored procedures or triggers there is a difference between client-side and server-side autocommit. From the client side, a stored procedure is a single statement, and so autocommit sends a single commit statement after the whole procedure is executed. From the database server perspective, the stored procedure may be composed of many SQL statements, and so server-side autocommit commits the results of each SQL statement within the procedure.

**Note**

Do not mix client-side and server-side implementations. You should not combine the setting of the chained option with the setting of the autocommit option in your SQL Anywhere ADO.NET, OLE DB, ODBC, PHP, or JDBC application.

## Isolation level settings

You can set the isolation level of a current connection using the `isolation_level` database option.

Some interfaces, such as ODBC, allow you to set the isolation level for a connection at connection time. You can reset this level later using the `isolation_level` database option. See [“isolation\\_level option” \[SQL Anywhere Server - Database Administration\]](#).

You can override any temporary or public settings for the `isolation_level` database option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

## Cursors and transactions

In general, a cursor closes when a COMMIT is performed. There are two exceptions to this behavior:

- The `close_on_endtrans` database option is set to Off.
- A cursor is opened WITH HOLD, which is the default with Open Client and JDBC.

If either of these two cases is true, the cursor remains open on a COMMIT.

### ROLLBACK and cursors

If a transaction rolls back, then cursors close except for those cursors opened WITH HOLD. However, don't rely on the contents of any cursor after a rollback.

The draft ISO SQL3 standard states that on a rollback, all cursors (even those cursors opened WITH HOLD) should close. You can obtain this behavior by setting the `ansi_close_cursors_on_rollback` option to On.

### Savepoints

If a transaction rolls back to a savepoint, and if the `ansi_close_cursors_on_rollback` option is On, then all cursors (even those cursors opened WITH HOLD) opened after the SAVEPOINT close.

### Cursors and isolation levels

You can change the isolation level of a connection during a transaction using the SET OPTION statement to alter the `isolation_level` option. However, this change does not affect open cursors.

A snapshot of all rows committed at the snapshot start time is visible when the WITH HOLD clause is used with the snapshot, statement-snapshot, and readonly-statement-snapshot isolation levels. Also visible are all modifications completed by the current connection since the start of the transaction within which the cursor was open. For more details about supported isolation levels, see [“Isolation levels and consistency” \[SQL Anywhere Server - SQL Usage\]](#) and [“isolation\\_level option” \[SQL Anywhere Server - Database Administration\]](#).



---

# .NET application programming

This section describes how to use SQL Anywhere with .NET, and includes the API for the SQL Anywhere .NET Data Provider.

## SQL Anywhere .NET Data Provider

This section describes SQL Anywhere Server .NET support, including the API for the SQL Anywhere .NET Data Provider.

## SQL Anywhere .NET support

ADO.NET is the latest data access API from Microsoft in the line of ODBC, OLE DB, and ADO. It is the preferred data access component for the Microsoft .NET Framework and allows you to access relational database systems.

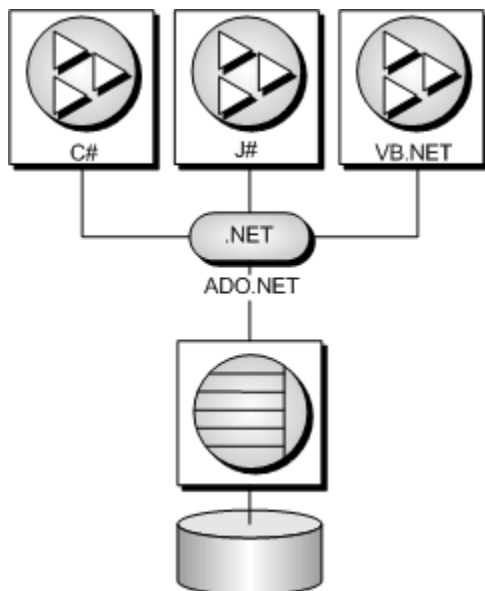
The SQL Anywhere .NET Data Provider implements the `iAnywhere.Data.SQLAnywhere` namespace and allows you to write programs in any of the .NET supported languages, such as C# and Visual Basic .NET, and access data from SQL Anywhere databases.

For general information about .NET data access, see the Microsoft ".NET Data Access Architecture Guide" at <http://msdn.microsoft.com/en-us/library/Ee817654%28pandp.10%29.aspx>.

### ADO.NET applications

You can develop Internet and intranet applications using object-oriented languages, and then connect these applications to SQL Anywhere using the ADO.NET data provider.

Combine this provider with built-in XML and web services features, .NET scripting capability for MobiLink synchronization, and an UltraLite.NET component for development of handheld database applications, and SQL Anywhere can integrate with the .NET Framework.



**See also**

- [“SQL Anywhere .NET Data Provider features” on page 40](#)
- [“Namespace” on page 99](#)
- [“Using the SQL Anywhere .NET Data Provider” on page 75](#)

## SQL Anywhere .NET Data Provider features

SQL Anywhere supports the Microsoft .NET Framework versions 2.0, 3.0, 3.5, and 4.0 through three distinct namespaces.

- **iAnywhere.Data.SQLAnywhere** The ADO.NET object model is an all-purpose data access model. ADO.NET components were designed to factor data access from data manipulation. There are two central components of ADO.NET that do this: the DataSet, and the .NET Framework data provider, which is a set of components including the Connection, Command, DataReader, and DataAdapter objects. SQL Anywhere includes a .NET Entity Framework Data Provider that communicates directly with a SQL Anywhere database server without adding the overhead of OLE DB or ODBC. The SQL Anywhere .NET Data Provider is represented in the .NET namespace as iAnywhere.Data.SQLAnywhere.

The Microsoft .NET Compact Framework is the smart device development framework for Microsoft .NET. The SQL Anywhere .NET Compact Framework Data Provider supports devices running Windows Mobile.

The SQL Anywhere .NET Data Provider namespace is described in this document.

To read more about how to access data stored inside a SQL Anywhere database using the ADO.NET object model, in particular via the Language Integrated Query (LINQ) to Entities methodology, see

the *SQL Anywhere and the ADO.NET Entity Framework* white paper at [www.sybase.com/detail?id=1060541](http://www.sybase.com/detail?id=1060541).

- **System.Data.OleDb** This namespace supports OLE DB data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use System.Data.OleDb together with the SQL Anywhere OLE DB provider, SAOLEDB, to access SQL Anywhere databases.
- **System.Data.Odbc** This namespace supports ODBC data sources. This namespace is an intrinsic part of the Microsoft .NET Framework. You can use System.Data.Odbc together with the SQL Anywhere ODBC driver to access SQL Anywhere databases.

On Windows Mobile, only the SQL Anywhere .NET Data Provider is supported.

There are some key benefits to using the SQL Anywhere .NET Data Provider:

- In the .NET environment, the SQL Anywhere .NET Data Provider provides native access to a SQL Anywhere database. Unlike the other supported providers, it communicates directly with a SQL Anywhere server and does not require bridge technology.
- As a result, the SQL Anywhere .NET Data Provider is faster than the OLE DB and ODBC Data Providers. It is the recommended Data Provider for accessing SQL Anywhere databases.

## .NET sample projects

There are several sample projects included with the SQL Anywhere .NET Data Provider:

- **LinqSample** A .NET Framework sample project for Windows that demonstrates language-integrated query, set, and transform operations using the SQL Anywhere .NET Data Provider and C#.
- **SimpleCE** A .NET Compact Framework sample project for Windows Mobile that demonstrates a simple listbox that is filled with the names from the Employees table when you click **Connect**.
- **SimpleWin32** A .NET Framework sample project for Windows that demonstrates a simple listbox that is filled with the names from the Employees table when you click **Connect**. For more information, see [“Tutorial: Using the Simple code sample” on page 76](#).
- **SimpleXML** A .NET Framework sample project for Windows that demonstrates how to obtain XML data from SQL Anywhere via ADO.NET. Samples for C#, Visual Basic, and Visual C++ are provided.
- **SimpleViewer** A .NET Framework sample project for Windows. For more information, see [“Tutorial: Developing a simple .NET database application with Visual Studio” on page 83](#).
- **TableViewer** A .NET Framework sample project for Windows that allows you to enter and execute SQL statements. For more information, see [“Tutorial: Using the Table Viewer code sample” on page 79](#).

## Using the .NET Data Provider in a Visual Studio project

Use the SQL Anywhere .NET Data Provider to develop .NET applications with Visual Studio by including both a reference to the SQL Anywhere .NET Data Provider, and a line in your source code referencing the SQL Anywhere .NET Data Provider classes.

### Prerequisites

There are no prerequisites for this task.

### Context and remarks

Many.

### Task

1. Start Visual Studio and open your project.
2. In the **Solution Explorer** window, right-click **References** and click **Add Reference**.

The reference indicates which provider to include and locates the code for the SQL Anywhere .NET Data Provider.

3. Click the **.NET** tab, and scroll through the list to locate any of the following:
  - iAnywhere.Data.SQLAnywhere for .NET 2
  - iAnywhere.Data.SQLAnywhere for .NET 3.5
  - iAnywhere.Data.SQLAnywhere for .NET 4

#### Note

There is separate version of the provider for Windows Mobile platforms. For the Windows Mobile SQL Anywhere .NET Data Provider, click the **Browse** tab and locate the Windows Mobile version of the provider. The default location is `%SQLANY12%\CE\Assembly\V2`.

4. Click the desired provider and then click **OK**.

The provider is added to the **References** folder in the **Solution Explorer** window of your project.

5. Specify a directive to your source code to assist with the use of the SQL Anywhere .NET Data Provider namespace and the defined types.

Add the following line to your project:

- If you are using C#, add the following line to the list of `using` directives at the beginning of your project:

```
using iAnywhere.Data.SQLAnywhere;
```

- If you are using Visual Basic, add the following line at the beginning of your project before the line `Public Class Form1`:

```
Imports iAnywhere.Data.SQLAnywhere
```

## Results

The SQL Anywhere .NET Data Provider is set up for use with your .NET application.

## Next

None.

## Example

The following example illustrates code that can be used to establish a database connection when a directive is specified:

```
SACConnection conn = new SACConnection();
```

The following example illustrates code that can be used to establish a database connection when a directive is not specified:

```
iAnywhere.Data.SQLAnywhere.SACConnection conn =  
    new iAnywhere.Data.SQLAnywhere.SACConnection();
```

## See also

- [“SQL Anywhere .NET Data Provider deployment” on page 69](#)

# Connecting to a database

Use C# code to connect to a SQL Anywhere database.

## Prerequisites

Create a Visual Studio application that implements the ADO.NET API.

## Context and remarks

Many.

## Task

1. Creates an SACConnection object named `conn` with the following code:

```
SACConnection conn = new SACConnection(connection-string);
```

You can have more than one connection to a database from your application. Some applications use a single connection to a SQL Anywhere database, and keep the connection open all the time. To do this, you can declare a global variable for the connection:

```
private SACConnection _conn;
```

For more information, see the sample code in %SQLANYSAMPI2%\SQLAnywhere\ADO.NET\TableView and [“Understanding the Table Viewer sample project” on page 81](#).

2. Specify the connection string used to connect to the database using the following code:

```
_conn.ConnectionString = "Data Source=SQL Anywhere 12 Demo"
```

Instead of supplying the full connection string, you can prompt users for their user ID and password parameters.

3. Open a connection to the database.

The following code attempts to connect to a database. It automatically starts the database server if necessary.

```
_conn.Open();
```

4. Catch connection errors.

Your application should be designed to catch any errors that occur when attempting to connect to the database. The following code demonstrates how to catch an error and display its message:

```
try
{
    _conn = new SAConnection( txtConnectString.Text );
    _conn.Open();
}
catch( SAException ex )
{
    MessageBox.Show( ex.Errors[0].Source + " : "
        + ex.Errors[0].Message + " (" +
        ex.Errors[0].NativeError.ToString() + ")",
        "Failed to connect" );
}
```

Alternately, you can use the `ConnectionString` property to set the connection string, rather than passing the connection string when the `SAConnection` object is created:

```
SAConnection _conn;
_conn = new SAConnection();
_conn.ConnectionString = txtConnectString.Text;
_conn.Open();
```

5. Close the connection to the database.

```
_conn.Close();
```

Connections to the database stay open until they are explicitly closed using the `Close` method.

## Results

The application opens a connection to the database.

## Next

Interface with the `SAConnection` object to perform database operations in your .NET application.

## Example

The following Visual Basic code creates a method that opens a connection to the SQL Anywhere sample database:

```
Private Sub Button1_Click(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) _
    Handles Button1.Click
    ' Declare the connection object
    Dim myConn As New iAnywhere.Data.SQLAnywhere.SAConnection()
    myConn.ConnectionString = "Data Source=SQL Anywhere 12 Demo"
    myConn.Open()
    myConn.Close()
End Sub
```

## See also

- “Connection parameters” [[SQL Anywhere Server - Database Administration](#)]
- “SAConnectionStringBuilder class [SQL Anywhere .NET]” on page 180
- “SAConnectionStringBuilder.ConnectionName property [SQL Anywhere .NET]” on page 188

## Connection pooling

The SQL Anywhere .NET Data Provider supports native .NET connection pooling. Connection pooling allows your application to reuse existing connections by saving the connection handle to a pool so it can be reused, rather than repeatedly creating a new connection to the database. Connection pooling is turned on by default.

The pool size is set in your connection string using the POOLING option. The default maximum pool size is 100, while the default minimum pool size is 0. You can specify the minimum and maximum pool sizes. For example:

```
"Data Source=SQL Anywhere 12 Demo;POOLING=TRUE;Max Pool Size=50;Min Pool
Size=5"
```

When your application first attempts to connect to the database, it checks the pool for an existing connection that uses the same connection parameters you have specified. If a matching connection is found, that connection is used. Otherwise, a new connection is used. When you disconnect, the connection is returned to the pool so that it can be reused.

The SQL Anywhere database server also supports connection pooling. This feature is controlled using the ConnectionPool (CPOOL) connection parameter. However, the SQL Anywhere .NET Data Provider does not use this server feature and disables it (CPOOL=NO). All connection pooling is done in the .NET client application instead (client-side connection pooling).

## See also

- “SAConnectionStringBuilder.ConnectionName property [SQL Anywhere .NET]” on page 188
- “AutoStop (ASTOP) connection parameter” [[SQL Anywhere Server - Database Administration](#)]

## Connection state

Once your application has established a connection to the database, you can check the connection state to ensure that the connection is open before you fetch data from the database to update it. If a connection is lost or is busy, or if another statement is being processed, you can return an appropriate message to the user.

The `SACConnection` class has a `State` property that can be used to check the state of the connection. Possible state values are `Open` and `Closed`.

The following code checks whether the `SACConnection` object has been initialized, and if it has, it ensures that the connection is open. A message is returned to the user if the connection is not open.

```
if ( _conn == null || _conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to a database first", "Not connected" );
    return;
}
```

### See also

- [“SACConnection.State property \[SQL Anywhere .NET\]” on page 179](#)

## Data access and manipulation

With the SQL Anywhere .NET Data Provider, there are two ways you can access data:

- **SACCommand object** The `SACCommand` object is the recommended way of accessing and manipulating data in .NET.

The `SACCommand` object allows you to execute SQL statements that retrieve or modify data directly from the database. Using the `SACCommand` object, you can issue SQL statements and call stored procedures directly against the database.

Within an `SACCommand` object, an `SADDataReader` is used to return read-only result sets from a query or stored procedure. The `SADDataReader` returns only one row at a time, but this does not degrade performance because the SQL Anywhere client-side libraries use prefetch buffering to prefetch several rows at a time.

Using the `SACCommand` object allows you to group your changes into transactions rather than operating in autocommit mode. When you use the `SATransaction` object, locks are placed on the rows so that other users cannot modify them.

For more information, see [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#) and [“SADDataReader class \[SQL Anywhere .NET\]” on page 214](#).

- **SADDataAdapter object** The `SADDataAdapter` object retrieves the entire result set into a `DataSet`. A `DataSet` is a disconnected store for data that is retrieved from a database. You can then edit the data in the `DataSet` and when you are finished, the `SADDataAdapter` object updates the database with the



changes made to the DataSet. When you use the SDataAdapter, there is no way to prevent other users from modifying the rows in your DataSet. You need to include logic within your application to resolve any conflicts that may occur.

For more information about conflicts, see [“Resolving conflicts when using the SDataAdapter” on page 53](#).

For more information about the SDataAdapter object, see [“SDataAdapter class \[SQL Anywhere .NET\]” on page 205](#).

There is no performance impact from using the SDataReader within an SACommand object to fetch rows from the database rather than the SDataAdapter object.

## Using the SACommand object to retrieve and manipulate data

The following sections describe how to retrieve data and how to insert, update, or delete rows using the SDataReader.

### Access data using the SACommand object

The SACommand object allows you to execute a SQL statement or call a stored procedure against a SQL Anywhere database. You can use any of the following methods to retrieve data from the database:

- **ExecuteReader** Issues a SQL query that returns a result set. This method uses a forward-only, read-only cursor. You can loop quickly through the rows of the result set in one direction.
- **ExecuteScalar** Issues a SQL query that returns a single value. This can be the first column in the first row of the result set, or a SQL statement that returns an aggregate value such as COUNT or AVG. This method uses a forward-only, read-only cursor.

When using the SACommand object, you can use the SDataReader to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

When using the SDataReader, there are several Get methods available that you can use to return the results in the specified data type.

### C# ExecuteReader example

The following C# code opens a connection to the SQL Anywhere sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 12 Demo");
conn.Open();
SACommand cmd = new SACommand("SELECT Surname FROM Employees", conn);
SDataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while (reader.Read())
```

```
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

### Visual Basic ExecuteReader example

The following Visual Basic code opens a connection to the SQL Anywhere sample database and uses the ExecuteReader method to create a result set containing the last names of employees in the Employees table:

```
Dim myConn As New SAConnection()
myConn.ConnectionString = "Data Source=SQL Anywhere 12 Demo"
Dim myCmd As New SACCommand("SELECT Surname FROM Employees", myConn)
Dim myReader As SADataReader

myConn.Open()
myReader = myCmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (myReader.Read())
    ListEmployees.Items.Add(myReader.GetString(0))
Loop
ListEmployees.EndUpdate()
myConn.Close()
```

### C# ExecuteScalar example

The following C# code opens a connection to the SQL Anywhere sample database and uses the ExecuteScalar method to obtain a count of the number of male employees in the Employees table:

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn );
int count = (int) cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

### See also

- [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)
- [“SACCommand.ExecuteReader method \[SQL Anywhere .NET\]” on page 137](#)
- [“SACCommand.ExecuteScalar method \[SQL Anywhere .NET\]” on page 139](#)

## Manipulate data using the SACCommand object

To perform an insert, update, or delete with the SACCommand object, use the ExecuteNonQuery function. The ExecuteNonQuery function issues a query (SQL statement or stored procedure) that does not return a result set.

You can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. You must be connected to a database to use the SACCommand object.

For information about obtaining primary key values for autoincrement primary keys, see [“Primary key values” on page 57](#).

If you want to set the isolation level for a SQL statement, you must use the SACommand object as part of an SATransaction object. When you modify data without an SATransaction object, the provider operates in autocommit mode and any changes that you make are applied immediately. See [“Transaction processing” on page 62](#).

### C# ExecuteNonQuery DELETE and INSERT example

The following example opens a connection to the SQL Anywhere sample database and uses the ExecuteNonQuery method to remove all departments whose ID is greater than or equal to 600 and then add two new rows to the Departments table. It displays the updated table in a datagrid.

```
SAConnection conn = new SAConnection("Data Source=SQL Anywhere 12 Demo");
conn.Open();

SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();

SACommand insertCmd = new SACommand(
    "INSERT INTO Departments(DepartmentID, DepartmentName) VALUES( ?, ? )",
    conn );
SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
insertCmd.Parameters.Add( parm );
parm = new SAParameter();
parm.SADbType = SADbType.Char;
insertCmd.Parameters.Add( parm );

insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();

insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

### C# ExecuteNonQuery UPDATE example

The following example opens a connection to the SQL Anywhere sample database and uses the ExecuteNonQuery method to update the DepartmentName column to "Engineering" in all rows of the Departments table where the DepartmentID is 100. It displays the updated table in a datagrid.

```
SACConnection conn = new SACConnection("Data Source=SQL Anywhere 12 Demo");
conn.Open();

SACCommand updateCmd = new SACCommand(
    "UPDATE Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();

SACCommand selectCmd = new SACCommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACCommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

### See also

- [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#)
- [“SACCommand.ExecuteNonQuery method \[SQL Anywhere .NET\]” on page 137](#)
- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)

## Result set schema information

You can obtain schema information about columns in a result set.

The GetSchemaTable method of the SADataReader class obtains information about the current result set. The GetSchemaTable method returns the standard .NET DataTable object, which provides information about all the columns in the result set, including column properties.

### C# schema information example

The following example obtains information about a result set using the GetSchemaTable method and binds the DataTable object to the datagrid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACCommand cmd = new SACCommand("SELECT * FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
DataTable schema = reader.GetSchemaTable();
reader.Close();
conn.Close();
dataGridView1.DataSource = schema;
```

### See also

- [“SADataReader.GetSchemaTable method \[SQL Anywhere .NET\]” on page 229](#)

## Using the SDataAdapter object to access and manipulate data

The following sections describe how to retrieve data and how to insert, update, or delete rows using the SDataAdapter.

### Access data and schema information using the SDataAdapter object

The SDataAdapter allows you to view the entire result set by using the Fill method to fill a DataTable with the results from a query by binding the DataTable to the display grid.

Using the SDataAdapter, you can pass any string (SQL statement or stored procedure) that returns a result set. When you use the SDataAdapter, all the rows are fetched in one operation using a forward-only, read-only cursor. Once all the rows in the result set have been read, the cursor is closed. The SDataAdapter allows you to make changes to the DataTable. Once your changes are complete, you must reconnect to the database to apply the changes.

You can use the SDataAdapter object to retrieve a result set that is based on a join. However, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins.

#### Caution

Any changes you make to the DataTable are made while you are disconnected from the database. This means that your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the DataTable are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

### C# SDataAdapter Fill example using a DataTable

The following example shows how to fill a DataTable using the SDataAdapter. It creates a new DataTable object named Results and a new SDataAdapter object. The SDataAdapter Fill method is used to fill the DataTable with the results of the query. The DataTable is then bound to the grid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
DataTable dt = new DataTable("Results");
SDataAdapter da = new SDataAdapter("SELECT * FROM Employees", conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

### C# SDataAdapter Fill example using a DataSet

The following example shows how to fill a DataTable using the SDataAdapter. It creates a new DataSet object and a new SDataAdapter object. The SDataAdapter Fill method is used to create a DataTable table named Results in the DataSet and then fill it with the results of the query. The Results DataTable is then bound to the grid on the screen.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
```

```
DataSet ds = new DataSet();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees", conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

### C# SADataAdapter FillSchema example using a DataTable

The following example shows how to use the FillSchema method to set up a new DataTable object with the same schema as a result set. The Additions DataTable is then bound to the grid on the screen.

```
SACONNECTION conn = new SACONNECTION( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SADATAADAPTER da = new SADATAADAPTER("SELECT * FROM Employees", conn);
DATATABLE schema = new DATATABLE("Additions");
da.FillSchema(schema, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = schema;
```

### C# SADataAdapter FillSchema example using a DataSet

The following example shows how to use the FillSchema method to set up a new DataTable object with the same schema as a result set. The DataTable is added to the DataSet using the Merge method. The Additions DataTable is then bound to the grid on the screen.

```
SACONNECTION conn = new SACONNECTION( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SADATAADAPTER da = new SADATAADAPTER("SELECT * FROM Employees", conn);
DATATABLE schema = new DATATABLE("Additions");
da.FillSchema(schema, SchemaType.Source);
DATASET ds = new DATASET();
ds.Merge(schema);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

### See also

- [“SADataAdapter class \[SQL Anywhere .NET\]” on page 205](#)

## Manipulate data using the SADataAdapter object

The SADataAdapter retrieves a result set into a DataTable. A DataSet is a collection of tables (DataTables) and the relationships and constraints between those tables. The DataSet is built into the .NET Framework, and is independent of the Data Provider used to connect to your database.

When you use the SADataAdapter, you must be connected to the database to fill a DataTable and to update the database with changes made to the DataTable. However, once the DataTable is filled, you can modify the DataTable while disconnected from the database.

If you do not want to apply your changes to the database right away, you can write the DataSet, including the data and/or the schema, to an XML file using the WriteXml method. Then, you can apply the changes at a later time by loading a DataSet with the ReadXml method. The following shows two examples.

```
ds.WriteXml("Employees.xml");
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

For more information, see the .NET Framework documentation for WriteXml and ReadXml.

When you call the Update method to apply changes from the DataSet to the database, the SDataAdapter analyzes the changes that have been made and then invokes the appropriate statements, INSERT, UPDATE, or DELETE, as necessary. When you use the DataSet, you can only make changes (inserts, updates, or deletes) to data that is from a single table. You cannot update result sets that are based on joins. If another user has a lock on the row you are trying to update, an exception is thrown.

**Caution**

Any changes you make to the DataSet are made while you are disconnected. This means that your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that may occur when changes from the DataSet are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

**Resolving conflicts when using the SDataAdapter**

When you use the SDataAdapter, no locks are placed on the rows in the database. This means there is the potential for conflicts to arise when you apply changes from the DataSet to the database. Your application should include logic to resolve or log conflicts that arise.

Some of the conflicts that your application logic should address include:

- **Unique primary keys** If two users insert new rows into a table, each row must have a unique primary key. For tables with AUTOINCREMENT primary keys, the values in the DataSet may become out of sync with the values in the data source.

For information about obtaining primary key values for AUTOINCREMENT primary keys, see [“Primary key values” on page 57](#).

- **Updates made to the same value** If two users modify the same value, your application should include logic to determine which value is correct.
- **Schema changes** If a user modifies the schema of a table you have updated in the DataSet, the update will fail when you apply the changes to the database.
- **Data concurrency** Concurrent applications should see a consistent set of data. The SDataAdapter does not place a lock on rows that it fetches, so another user can update a value in the database once you have retrieved the DataSet and are working offline.

Many of these potential problems can be avoided by using the SACommand, SDataReader, and SATransaction objects to apply changes to the database. The SATransaction object is recommended because it allows you to set the isolation level for the transaction and it places locks on the rows so that other users cannot modify them.

For more information about using transactions to apply your changes to the database, see [“Manipulate data using the SACommand object” on page 48](#).

To simplify the process of conflict resolution, you can design your INSERT, UPDATE, or DELETE statement to be a stored procedure call. By including INSERT, UPDATE, and DELETE statements in stored procedures, you can catch the error if the operation fails. In addition to the statement, you can add

error handling logic to the stored procedure so that if the operation fails the appropriate action is taken, such as recording the error to a log file, or trying the operation again.

## Insert rows into a table using an **SADDataAdapter** object

### **C# SADDataAdapter** Insert example

The following example shows how to use the Update method of SADDataAdapter to add rows to a table. The example fetches the Departments table into a DataTable using the SelectCommand property and the Fill method of the SADDataAdapter. It then adds two new rows to the DataTable and updates the Departments table from the DataTable using the InsertCommand property and the Update method of the SADDataAdapter.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();

SADDataAdapter da = new SADDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.InsertCommand = new SACommand(
    "INSERT INTO Departments( DepartmentID, DepartmentName ) " +
    "VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );

DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );

rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```



**See also**

- [“SDataAdapter class \[SQL Anywhere .NET\]” on page 205](#)

**Update rows in a table using an SDataAdapter object****C# SDataAdapter Update example**

The following example shows how to use the Update method of SDataAdapter to update rows in a table. The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SDataAdapter. It then modifies some values in the DataTable and updates the Departments table from the DataTable using the UpdateCommand property and the Update method of the SDataAdapter.

```

SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.UpdateCommand = new SACommand(
    "UPDATE Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );

```

```
dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

### See also

- [“SDataAdapter class \[SQL Anywhere .NET\]” on page 205](#)

### Delete rows from a table using an SDataAdapter object

#### C# SDataAdapter Delete example

The following example shows how to use the Update method of SDataAdapter to delete rows from a table. The example adds two new rows to the Departments table and then fetches this table into a DataTable using the SelectCommand property and the Fill method of the SDataAdapter. It then deletes some rows from the DataTable and updates the Departments table from the DataTable using the DeleteCommand property and the Update method of the SDataAdapter.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACCommand prepCmd = new SACCommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new SACCommand(
    "SELECT * FROM Departments", conn);
da.DeleteCommand = new SACCommand(
    "DELETE FROM Departments WHERE DepartmentID = ?",
    conn);
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add(parm);

DataTable dataTable = new DataTable("Departments");
int rowCount = da.Fill(dataTable);

foreach (DataRow row in dataTable.Rows)
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row.Delete();
    }
}
rowCount = da.Update(dataTable);

dataTable.Clear();
```

```

rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;

```

**See also**

- [“SADDataAdapter class \[SQL Anywhere .NET\]” on page 205](#)

## Primary key values

If the table you are updating has an autoincremented primary key, uses UUIDs, or if the primary key comes from a primary key pool, you can use a stored procedure to obtain values generated by the data source.

### Insert a new row with an autoincremented primary key using an SACommand object

#### C# SACommand primary key example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SACommand object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet\_primarykey) and the stored procedure (sp\_adodotnet\_primarykey) that will be used to insert rows and return primary key values.

```

SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ( " +
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE sp_adodotnet_primarykey( " +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parmName);

```

```
parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id1);

parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id2);

parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id3);

parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id4);

cmd.CommandText = "SELECT * FROM adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
SADataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
    System.Console.WriteLine(dr.GetString(0) + ", " +
        dr.GetString(1));
}
dr.Close();
conn.Close();
```

#### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)
- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)

## Insert a new row with an autoincremented primary key using an SADDataAdapter object

### C# SADDataAdapter primary key example

The following example shows how to obtain the primary key that is generated for a newly inserted row. The example uses an SADDataAdapter object to call a SQL stored procedure and an SAParameter object to retrieve the primary key that it returns. For demonstration purposes, the example creates a sample table (adodotnet\_primarykey) and the stored procedure (sp\_adodotnet\_primarykey) that will be used to insert rows and return primary key values.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey (" +
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();
```

```
cmd.CommandText = "CREATE or REPLACE PROCEDURE sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

SDataAdapter da = new SDataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

da.SelectCommand = new SACommand(
    "SELECT * FROM adodotnet_primarykey", conn);

da.InsertCommand = new SACommand(
    "sp_adodotnet_primarykey", conn);
da.InsertCommand.CommandType = CommandType.StoredProcedure;
da.InsertCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "Name";
parmName.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmName);

DataTable dataTable = new DataTable("Departments");
da.FillSchema(dataTable, SchemaType.Source);

DataRow row = dataTable.NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataTable.Rows.Add(row);

DataSet ds = new DataSet();
ds.Merge(dataTable);
da.Update(ds, "Departments");

conn.Close();
dataGridView1.DataSource = ds.Tables["Departments"];
```

### See also

- [“SADataAdapter class \[SQL Anywhere .NET\]” on page 205](#)
- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)

## BLOBs

When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the `GetBytes` method, and for string data, use the `GetChars` method. Otherwise, BLOB data is treated in the same manner as any other data you fetch from the database.

### C# GetChars BLOB example

The following example reads three columns from a result set. The first two columns are integers, while the third column is a LONG VARCHAR. The length of the third column is computed by reading this column with the `GetChars` method in chunks of 100 characters.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM MarketingInformation", conn);
SADataReader reader = cmd.ExecuteReader();

int idValue;
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0, length))
        == (long)length)
    {
        blobLength += charsRead;
    }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

### See also

- [“SADataReader.GetBytes method \[SQL Anywhere .NET\]” on page 219](#)
- [“SADataReader.GetChars method \[SQL Anywhere .NET\]” on page 220](#)

## Time values

The .NET Framework does not have a Time structure. If you want to fetch time values from SQL Anywhere, you must use the `GetTimeSpan` method. This method returns the data as a .NET Framework `TimeSpan` object.

## C# TimeSpan example

The following example uses the `GetTimeSpan` method to return the time as `TimeSpan`.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT 123, CURRENT TIME", conn);
SADataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    int ID = reader.GetInt32(0);
    TimeSpan time = reader.GetTimeSpan(1);
}
reader.Close();
conn.Close();
```

### See also

- [“SADaReader.GetTimeSpan method \[SQL Anywhere .NET\]” on page 231](#)

## Stored procedures

You can use SQL stored procedures with the SQL Anywhere .NET Data Provider.

The `ExecuteReader` method is used to call stored procedures that return result sets, while the `ExecuteNonQuery` method is used to call stored procedures that do not return any result sets. The `ExecuteScalar` method is used to call stored procedures that return only a single value.

You can use `SAParameter` objects to pass parameters to a stored procedure.

### C# stored procedure call with parameters example

The following example shows two ways to call a stored procedure and pass it a parameter. The example uses an `SADaReader` to fetch the result set returned by the stored procedure.

```
SAConnection conn = new SAConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
bool method1 = true;

SACommand cmd = new SACommand("", conn);
if (method1)
{
    cmd.CommandText = "ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
{
    cmd.CommandText = "call ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}

SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);

SADaReader reader = cmd.ExecuteReader();
```

```
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
reader.Close();

listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();

conn.Close();
```

### See also

- [“SACCommand.ExecuteReader method \[SQL Anywhere .NET\]” on page 137](#)
- [“SACCommand.ExecuteNonQuery method \[SQL Anywhere .NET\]” on page 137](#)
- [“SACCommand.ExecuteScalar method \[SQL Anywhere .NET\]” on page 139](#)
- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“Manipulate data using the SACCommand object” on page 48](#)
- [“Access data using the SACCommand object” on page 47](#)

## Transaction processing

With the SQL Anywhere .NET Data Provider, you can use the `SATransaction` object to group statements together. Each transaction ends with a `COMMIT` or `ROLLBACK`, which either makes your changes to the database permanent or cancels all the operations in the transaction. Once the transaction is complete, you must create a new `SATransaction` object to make further changes. This behavior is different from ODBC and embedded SQL, where a transaction persists after you execute a `COMMIT` or `ROLLBACK` until the transaction is closed.

If you do not create a transaction, the SQL Anywhere .NET Data Provider operates in autocommit mode by default. There is an implicit `COMMIT` after each insert, update, or delete, and once an operation is completed, the change is made to the database. In this case, the changes cannot be rolled back.

### Isolation level settings for transactions

The database isolation level is used by default for transactions. You can choose to specify the isolation level for a transaction using the `IsolationLevel` property when you begin the transaction. The isolation level applies to all statements executed within the transaction. The SQL Anywhere .NET Data Provider supports snapshot isolation.

The locks that SQL Anywhere uses when you execute a SQL statement depend on the transaction's isolation level.

### Distributed transaction processing

The .NET 2.0 framework introduced a new namespace `System.Transactions`, which contains classes for writing transactional applications. Client applications can create and participate in distributed transactions with one or multiple participants. Client applications can implicitly create transactions using the `TransactionScope` class. The connection object can detect the existence of an ambient transaction created



by the `TransactionScope` and automatically enlist. The client applications can also create a `CommittableTransaction` and call the `EnlistTransaction` method to enlist. This feature is supported by the SQL Anywhere .NET Data Provider. Distributed transaction has significant performance overhead. It is recommended that you use database transactions for non-distributed transactions.

### C# SATransaction example

The following example shows how to wrap an INSERT into a transaction so that it can be committed or rolled back. A transaction is created with an `SATransaction` object and linked to the execution of a SQL statement using an `SACCommand` object. Isolation level 2 (`RepeatableRead`) is specified so that other database users cannot update the row. The lock on the row is released when the transaction is committed or rolled back. If you do not use a transaction, the SQL Anywhere .NET Data Provider operates in autocommit mode and you cannot roll back any changes that you make to the database.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12 Demo" );
conn.Open();
string stmt = "UPDATE Products SET UnitPrice = 2000.00 " +
              "WHERE Name = 'Tee shirt'";
bool goAhead = false;

SATransaction trans = conn.BeginTransaction(IsolationLevel.RepeatableRead);
SACCommand cmd = new SACCommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
    trans.Commit();
else
    trans.Rollback();
conn.Close();
```

### See also

- [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#)
- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)
- [“Isolation levels and consistency” \[SQL Anywhere Server - SQL Usage\]](#)
- [“Locks during queries” \[SQL Anywhere Server - SQL Usage\]](#)

## Error handling and the SQL Anywhere .NET Data Provider

Your application must be designed to handle any errors that occur, including ADO.NET errors. ADO.NET errors are handled within your code in the same way that you handle other errors in your application.

The SQL Anywhere .NET Data Provider creates an `SAException` object and throws an exception whenever errors occur during execution. Each `SAException` object consists of a list of `SAError` objects, and these error objects include the error message and code.

Errors are different from conflicts. Conflicts arise when changes are applied to the database. Your application should include a process to compute correct values or to log conflicts when they arise.

## C# Error handling example

The following example is from the Table Viewer sample project. If there is an error when the application attempts to connect to the database, the following code uses a try and catch block to catch the error and display its message:

```
try
{
    _conn = new SAConnection( txtConnectionString.Text );
    _conn.Open();
}
catch( SAException ex )
{
    MessageBox.Show( ex.Errors[0].Source + " : "
        + ex.Errors[0].Message + " (" +
        ex.Errors[0].NativeError.ToString() + ")",
        "Failed to connect" );
}
```

### See also

- [“SAError class \[SQL Anywhere .NET\]” on page 242](#)
- [“SAException class \[SQL Anywhere .NET\]” on page 246](#)
- [“SAFactory class \[SQL Anywhere .NET\]” on page 249](#)
- [“Resolving conflicts when using the SADataAdapter” on page 53](#)
- [“Understanding the Simple sample project” on page 77](#)
- [“Understanding the Table Viewer sample project” on page 81](#)

## Entity Framework support

The SQL Anywhere .NET Data Provider supports Entity Framework 4.2, a separate package available from Microsoft. To use Entity Framework 4.2, you must add it to Visual Studio using Microsoft's NuGet Package Manager.

One of the new features of Entity Framework is Code First. It enables a different development workflow: defining data model objects by simply writing C# or VB.NET classes mapping to database objects without ever having to open a designer or define an XML mapping file. Optionally, additional configuration can be performed by using data annotations or the Fluent API. Models can be used to generate a database schema or map to an existing database.

Here's an example which creates new database objects using the model:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "EdmCategories", Schema = "DBA" )]
    public class Category
    {
    }
```

```

        public string CategoryID { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }

        public virtual ICollection<Product> Products { get; set; }
    }

    [Table( "EdmProducts", Schema = "DBA" )]
    public class Product
    {
        public int ProductId { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
        public string CategoryID { get; set; }

        public virtual Category Category { get; set; }
    }

    [Table( "EdmSuppliers", Schema = "DBA" )]
    public class Supplier
    {
        [Key]
        public string SupplierCode { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Supplier>().Property( s =>
s.Name ).IsRequired();
        }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new SACConnectionFactory();
            Database.SetInitializer<Context>( new
DropCreateDatabaseAlways<Context>() );

            using ( var db = new Context( "DSN=SQL Anywhere 12 Demo" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}

```

To build and run this example, the following assembly references must be added:

```
EntityFramework
iAnywhere.Data.SQLAnywhere.v4.0
System.ComponentModel.DataAnnotations
System.Data.Entity
```

Here is another example that maps to an existing database:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "Customers", Schema = "GROUPO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }

    [Table( "Contacts", Schema = "GROUPO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
    }
}
```

```

    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new SAConnectionFactory();
            Database.SetInitializer<Context>( null );

            using ( var db = new Context( "DSN=SQL Anywhere 12 Demo" ) )
            {
                foreach ( var customer in db.Customers.ToList() )
                {
                    Console.WriteLine( "Customer - " + string.Format(
                        "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}",
                        customer.ID, customer.SurName, customer.GivenName,
                        customer.Street, customer.City, customer.State,
                        customer.Country, customer.PostalCode,
                        customer.Phone, customer.CompanyName ) );

                    foreach ( var contact in customer.Contacts )
                    {
                        Console.WriteLine( "    Contact - " + string.Format(
                            "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}, {10}",
                            contact.ID, contact.SurName, contact.GivenName,
                            contact.Title,
                            contact.Street, contact.City, contact.State,
                            contact.Country, contact.PostalCode,
                            contact.Phone, contact.Fax ) );
                    }
                }
            }
        }
    }
}

```

There are some implementation detail differences between the Microsoft SQL Server SqlClient provider and the SQL Anywhere .NET provider of which you should be aware.

1. A new class `SAConnectionFactory` (implements `IDbConnectionFactory`) is included. You set the `Database.DefaultConnectionFactory` to an instance of `SAConnectionFactory` before creating any data model as shown below:

```
Database.DefaultConnectionFactory = new SAConnectionFactory();
```

2. The major principle of Entity Framework Code First is coding by conventions. The Entity Framework infers the data model by coding conventions. Entity Framework also does lots of things implicitly. Sometimes the developer might not realize all these Entity Framework conventions. But some code conventions do not make sense for SQL Anywhere. There is a big difference between SQL Server and SQL Anywhere. Every SQL Server instance maintains multiple databases, but every SQL Anywhere database is a single file.
  - If the user creates a user-defined `DbContext` using the parameterless constructor, `SqlClient` will connect to SQL Server Express on the local machine using integrated security. The SQL Anywhere provider connects to the default server using integrated login if the user has already created a login mapping.

- SqlClient drops existing database and creates a new database when EF calls DbDeleteDatabase or DbCreateDatabase (SQL Server Express Edition only). The SQL Anywhere provider never drops or creates the database. It creates or drops the database objects (tables, relations, constraints for example). The user must create the database first.
  - The IDbConnectionFactory.CreateConnection method treats the string parameter "nameOrConnectionString" as database name (initial catalog for SQL Server) or a connection string. If the user does not provide the connection string for DbContext, SqlClient will automatically connect to the SQL Express server on the local machine using the namespace of user-defined DbContext class as the initial catalog. For SQL Anywhere, that parameter can only contain a connection string. A database name will be ignored and integrated login will be used instead.
3. The SQL Server SqlClient API maps a column with data annotation attribute TimeStamp to SQL Server data type timestamp/rowversion. There are some misconceptions about SQL Server timestamp/rowversion among developers. The SQL Server timestamp/rowversion data type is different from SQL Anywhere and most other database vendors:
- The SQL Server timestamp/rowversion is binary(8). It does not support a combined date and time value. SQL Anywhere supports a data type called timestamp that is equivalent to the SQL Server datetime data type.
  - SQL Server timestamp/rowversion values are guaranteed to be unique. SQL Anywhere timestamp values are not unique.
  - A SQL Server timestamp/rowversion value changes every time the row is updated.

The TimeStamp data annotation attribute is not supported by SQL Anywhere provider.

4. By default, Entity Framework 4.1 always sets the schema or owner name to **dbo** which is the default schema of SQL Server. However, **dbo** is not appropriate for SQL Anywhere. For SQL Anywhere, you must specify the schema name (**GROUPO** for example) with the table name either by using data annotations or the Fluent API. Here's an example:

```
namespace CodeFirstTest
{
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }

    public class Contact
    {
```

```

        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    [Table( "Departments", Schema = "GROUPO" )]
    public class Department
    {
        [Key()]
        public int DepartmentID { get; set; }
        public string DepartmentName { get; set; }
        public int DepartmentHeadID { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Department> Departments { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Contact>().ToTable( "Contacts",
"GROUPO" );
            modelBuilder.Entity<Customer>().ToTable( "Customers",
"GROUPO" );
        }
    }
}

```

## SQL Anywhere .NET Data Provider deployment

The following sections describe how to deploy the SQL Anywhere .NET Data Provider.

### SQL Anywhere .NET Data Provider system requirements

To use the SQL Anywhere .NET Data Provider, you must have the following installed on your computer or handheld device:

- The .NET Framework and/or .NET Compact Framework version 2.0 or later.

- Visual Studio 2005 or later, or a .NET language compiler, such as C# (required only for development).

## SQL Anywhere .NET Data Provider required files

The SQL Anywhere .NET Data Provider code resides in a DLL for each platform.

### Windows required files

For Windows (except Windows Mobile), one of the following DLLs is required:

- *%SQLANY12%\V2\Assembly\iAnywhere.Data.SQLAnywhere.dll*
- *%SQLANY12%\V2\Assembly\V3.5\iAnywhere.Data.SQLAnywhere.v3.5.dll*
- *%SQLANY12%\V2\Assembly\V4\iAnywhere.Data.SQLAnywhere.v4.0.dll*

The choice of DLL depends on the version of .NET that you are targeting.

The Windows version of the provider also requires the following DLLs.

- ***policy.12.0.iAnywhere.Data.SQLAnywhere.dll*** The policy file can be used to override the provider version that the application was built with. The policy file is updated by Sybase whenever an update to the provider is released. There are also policy files for the version 3.5 provider (*policy.12.0.iAnywhere.Data.SQLAnywhere.v3.5.dll*) and the version 4.0 provider (*policy.12.0.iAnywhere.Data.SQLAnywhere.v4.0.dll*).
- ***dbngen12.dll*** This language DLL contains English (en) messages issued by the provider. It is available in many other languages including Chinese (zh), French (fr), German (de), and Japanese (jp).
- ***dbcon12.dll*** The **Connect to SQL Anywhere** window support code is contained in this DLL.

### Windows Mobile required files

For Windows Mobile, the following DLLs are required:

- ***%SQLANY12%\CE\Assembly\iAnywhere.Data.SQLAnywhere.dll*** This is the Windows Mobile version of the provider.
- ***%SQLANY12%\CE\Arm.50\dbngen12.dll*** This language DLL contains English (en) messages issued by the provider. It is available in many other languages including Chinese (zh), French (fr), German (de), and Japanese (jp).
- ***%SQLANY12%\CE\Arm.50\dbcon12.dll*** The **Connect to SQL Anywhere** window support code is contained in this DLL.

The Windows Mobile version of the DLL is required for .NET Compact Framework version 2.0 or later applications.

Visual Studio deploys the .NET Data Provider DLL (*iAnywhere.Data.SQLAnywhere.dll*) to your device along with your program. If you are not using Visual Studio, you must copy the Data Provider DLL to the



device along with your application. It can go in the same directory as your application, or in the `\Windows` directory.

## The SQL Anywhere .NET Data Provider dbdata DLL

When the SQL Anywhere .NET Data Provider is first loaded by a .NET application (usually when making a database connection using `SACConnection`), it unpacks a DLL that contains the provider's unmanaged code. The file `dbdata12.dll` is placed by the provider in a subdirectory of the directory identified using the following strategy.

1. The first directory it attempts to use for unloading is the one returned by the first of the following:
  - The path identified by the **TMP** environment variable.
  - The path identified by the **TEMP** environment variable.
  - The path identified by the **USERPROFILE** environment variable.
  - The Windows directory.
2. If the identified directory is inaccessible, then the provider will attempt to use the current working directory.
3. If the current working directory is inaccessible, then the provider will attempt to use the directory from where the application itself was loaded.

The subdirectory name will take the form of a GUID with a suffix including the version number, bitness of the DLL, and an index number used to guarantee uniqueness. The following is an example of a possible subdirectory name.

```
{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_1201.x64_1
```

## SQL Anywhere .NET Data Provider DLL registration

The Windows version of the SQL Anywhere .NET Data Provider DLL (`%SQLANY12%\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll`) is registered in the Global Assembly Cache when you install the SQL Anywhere software. On Windows Mobile, you do not need to register the DLL.

If you are deploying the SQL Anywhere .NET Data Provider, you can register it using the **gacutil** utility that is included with the Microsoft SDK.

To register the SQL Anywhere .NET Data Provider as a `DbProviderFactory` instance when deploying the provider, you must add an entry to the `.NET machine.config` file. An entry similar to the following must be placed in the `<DbProviderFactories>` section.

```
<add invariant="iAnywhere.Data.SQLAnywhere"
name="SQL Anywhere 12 Data Provider"
description=".Net Framework Data Provider for SQL Anywhere 12"
type="iAnywhere.Data.SQLAnywhere.SAFactory, iAnywhere.Data.SQLAnywhere.v3.5,
Version=12.0.1.36003, Culture=neutral, PublicKeyToken=f222fc4333e0d400"/>
```

The version number must match the version of the provider that you are installing. The configuration file is located in `\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`. For 64-bit Windows systems, there is a second configuration file under the `Framework64` tree that must also be modified.

## .NET tracing support

The SQL Anywhere .NET provider supports tracing using the .NET tracing feature. Note that tracing is not supported on Windows Mobile.

By default, tracing is disabled. To enable tracing, specify the trace source in your application's configuration file.

The following is an example of a configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
  <source name="iAnywhere.Data.SQLAnywhere"
    switchName="SASourceSwitch"
    switchType="System.Diagnostics.SourceSwitch">
    <listeners>
      <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
      <add name="EventListener"
        type="System.Diagnostics.EventLogTraceListener"
        initializeData="MyEventLog"/>
      <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
      <remove name="Default"/>
    </listeners>
  </source>
</sources>
<switches>
  <add name="SASourceSwitch" value="All"/>
  <add name="SATraceAllSwitch" value="1" />
  <add name="SATraceExceptionSwitch" value="1" />
  <add name="SATraceFunctionSwitch" value="1" />
  <add name="SATracePoolingSwitch" value="1" />
  <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

There are four types of trace listeners referenced in the configuration file shown above.

- **ConsoleTraceListener** Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the **Output** window.
- **DefaultTraceListener** This listener is automatically added to the `Debug.Listeners` and `Trace.Listeners` collections using the name "Default". Tracing or debugging output is directed to either the standard output or the standard error stream. When using Microsoft Visual Studio, output appears in the **Output** window. To avoid duplication of output produced by the **ConsoleTraceListener**, this listener is removed.

- **EventLogTraceListener** Tracing or debugging output is directed to an **EventLog** identified in the **initializeData** option. In the example, the event log is named **MyEventLog**. Writing to the system event log requires administrator privileges and is not a recommended method for debugging applications.
- **TextWriterTraceListener** Tracing or debugging output is directed to a **TextWriter** which writes the stream to the file identified in the **initializeData** option.

To disable tracing to any of the trace listeners described above, remove the corresponding **add** entry under **<listeners>**.

The trace configuration information is placed in the application's project folder in the *App.config* file. If the file does not exist, it can be created and added to the project using Visual Studio by choosing **Add » New Item** and selecting **Application Configuration File**.

The **traceOutputOptions** can be specified for any listener and include the following:

- **Callstack** Write the call stack, which is represented by the return value of the `Environment.StackTrace` property.
- **DateTime** Write the date and time.
- **LogicalOperationStack** Write the logical operation stack, which is represented by the return value of the `CorrelationManager.LogicalOperationStack` property.
- **None** Do not write any elements.
- **ProcessId** Write the process identity, which is represented by the return value of the `Process.Id` property.
- **ThreadId** Write the thread identity, which is represented by the return value of the `Thread.ManagedThreadId` property for the current thread.
- **Timestamp** Write the timestamp, which is represented by the return value of the `System.Diagnostics.Stopwatch.GetTimeStamp` method.

The example configuration file, shown earlier, specifies trace output options for the **TextWriterTraceListener** only.

You can limit what is traced by setting specific trace options. By default the trace option settings are all 0. The trace options that can be set include the following:

- **SATraceAllSwitch** All the trace options are enabled. You do not need to set any other options since they are all selected. You cannot disable individual options if you choose this option. For example, the following will not disable exception tracing.

```
<add name="SATraceAllSwitch" value="1" />
<add name="SATraceExceptionSwitch" value="0" />
```

- **SATraceExceptionSwitch** All exceptions are logged. Trace messages have the following form.

```
<Type|ERR> message='message_text'[ nativeError=error_number]
```

The `nativeError=error_number` text will only be displayed if there is an `SAException` object.

- **SATraceFunctionSwitch** All function scope entry/exits are logged. Trace messages have any of the following forms.

```
enter_nnn <sa.class_name.method_name|API> [object_id#][parameter_names]
leave_nnn
```

The `nnn` is an integer representing the scope nesting level 1, 2, 3, ... The optional `parameter_names` is a list of parameter names separated by spaces.

- **SATracePoolingSwitch** All connection pooling is logged. Trace messages have any of the following forms.

```
<sa.ConnectionPool.AllocateConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.RemoveConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReturnConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReuseConnection|CPOOL>
connectionString='connection_text'
```

- **SATracePropertySwitch** All property setting and retrieval is logged. Trace messages have any of the following forms.

```
<sa.class_name.get_property_name|API> object_id#
<sa.class_name.set_property_name|API> object_id#
```

For more information, see "Tracing Data Access" at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnadonet/html/tracingdataaccess.asp>.

## Configuring a Windows application for tracing

Enabling tracing on the **TableViewer** sample application involves creating a configuration file that references the **ConsoleTraceListener** and **TextWriterTraceListener** listeners, removes the default listener, and enables all switches that would otherwise be set to 0.

### Prerequisites

You must have Visual Studio installed.

### Context and remarks

Windows Mobile does not support tracing.

### Task

1. Open the **TableViewer** sample in Visual Studio.

Start Visual Studio and open the `%SQLANYXSAMP12%\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln`.

2. Create an application file named *App.config* and copy the following configuration setup:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
  <source name="iAnywhere.Data.SQLAnywhere"
    switchName="SASourceSwitch"
    switchType="System.Diagnostics.SourceSwitch">
    <listeners>
      <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
      <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
      <remove name="Default"/>
    </listeners>
    </source>
  </sources>
<switches>
  <add name="SASourceSwitch" value="All"/>
  <add name="SATraceAllSwitch" value="1" />
  <add name="SATraceExceptionSwitch" value="1" />
  <add name="SATraceFunctionSwitch" value="1" />
  <add name="SATracePoolingSwitch" value="1" />
  <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

3. Rebuild the application.
4. Click **Debug** » **Start Debugging**.

## Results

When the application finishes execution, the trace output is recorded in the *bin\Debug\myTrace.log* file.

## Next

View the trace log in the **Output** window of Visual Studio.

## See also

- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnadonet/html/tracingdataaccess.asp>

# Using the SQL Anywhere .NET Data Provider

This section explains how to use the Simple and Table Viewer sample projects that are included with the SQL Anywhere .NET Data Provider. The sample projects can be used with Visual Studio 2005 or later versions. The sample projects were developed with Visual Studio 2005. If you use a later version, you may have to run the Visual Studio **Upgrade Wizard**. This section also includes a tutorial that takes you through the steps of building the Simple Viewer .NET database application using Visual Studio.

## Tutorial: Using the Simple code sample

The Simple project uses the .NET Data Provider to obtain a result set from the database server.

### Prerequisites

Microsoft Visual Studio

### Context and remarks

The Simple project is included with the SQL Anywhere samples. It demonstrates a simple listbox that is filled with the names from the Employees table.

### Run the Simple code sample in Visual Studio

1. Start Visual Studio.
2. Click **File » Open » Project**.
3. Browse to `%SQLANYSAMPI2%\SQLAnywhere\ADO.NET\SimpleWin32` and open the *Simple.sln* project.
4. When you use the SQL Anywhere .NET Data Provider in a project, you must add a reference to the Data Provider. This has already been done in the Simple code sample. To view the reference to the Data Provider (*iAnywhere.Data.SQLAnywhere*), open the **References** folder in the **Solution Explorer** window.
5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Simple code sample. To view the `using` directive:
  - Open the source code for the project. In the **Solution Explorer** window, right-click *Form1.cs* and click **View Code**.

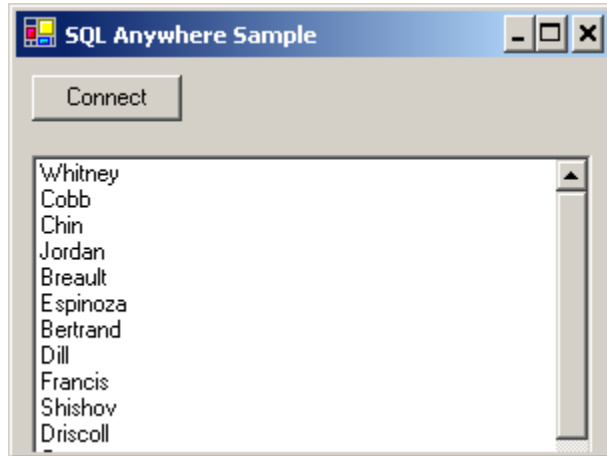
In the `using` directives in the top section, you should see the following line:

```
using iAnywhere.Data.SQLAnywhere;
```

This line is required for C# projects. If you are using Visual Basic .NET, you need to add an `Imports` line to your source code.

6. Click **Debug » Start Without Debugging** or press Ctrl+F5 to run the Simple sample.
7. In the **SQL Anywhere Sample** window, click **Connect**.

The application connects to the SQL Anywhere sample database and puts the surname of each employee in the window, as follows:



8. Close the **SQL Anywhere Sample** window to shut down the application and disconnect from the sample database. This also shuts down the database server.

## Results

You have built and executed a simple .NET application that uses the SQL Anywhere .NET Data Provider to obtain a result set from a SQL Anywhere database server.

## Next

None.

## Example

The complete application can be found in the SQL Anywhere samples directory at `%SQLANYSAMPI2%\SQLAnywhere\ADO.NET\SimpleWin32`.

## See also

- [“Using the .NET Data Provider in a Visual Studio project”](#) on page 42
- [“Understanding the Simple sample project”](#) on page 77

## Understanding the Simple sample project

This section illustrates some key features of the SQL Anywhere .NET Data Provider by walking through some of the code from the Simple code sample. The Simple code sample uses the SQL Anywhere sample database, *demo.db*, which is held in your SQL Anywhere samples directory.

For information about the location of the SQL Anywhere samples directory, see [“Samples directory”](#) [*SQL Anywhere Server - Database Administration*].

For information about the sample database, including the tables in the database and the relationships between them, see [“SQL Anywhere sample database”](#) [*SQL Anywhere 12 - Introduction*].

In this section, the code is described a few lines at a time. Not all code from the sample is included here. To see all the code, open the sample project in `%SQLANYSAMPI2%\SQLAnywhere\ADO.NET\SimpleWin32`.

## Declaring controls

The following code declares a button named `btnConnect` and a listbox named `listEmployees`.

```
private System.Windows.Forms.Button btnConnect;  
private System.Windows.Forms.ListBox listEmployees;
```

## Connecting to the database

The `btnConnect_Click` method declares and initializes an `SACConnection` connection object.

```
SACConnection conn = new SACConnection( "Data Source=SQL Anywhere 12  
Demo;UID=DBA;PWD=sql" );
```

The `SACConnection` object uses the connection string to connect to the SQL Anywhere sample database when the `Open` method is called.

```
conn.Open();
```

For more information about the `SACConnection` object, see [“SACConnection class \[SQL Anywhere .NET\]” on page 160](#).

## Defining a query

A SQL statement is executed using an `SACCommand` object. The following code declares and creates a command object using the `SACCommand` constructor. This constructor accepts a string representing the query to be executed, along with the `SACConnection` object that represents the connection that the query is executed on.

```
SACCommand cmd = new SACCommand( "SELECT Surname FROM Employees", conn );
```

For more information about the `SACCommand` object, see [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#).

## Displaying the results

The results of the query are obtained using an `SADataReader` object. The following code declares and creates an `SADataReader` object using the `ExecuteReader` constructor. This constructor is a member of the `SACCommand` object, `cmd`, that was declared previously. `ExecuteReader` sends the command text to the connection for execution and builds an `SADataReader`.

```
SADataReader reader = cmd.ExecuteReader();
```

The following code loops through the rows held in the `SADataReader` object and adds them to the listbox control. Each time the `Read` method is called, the data reader gets another row back from the result set. A new item is added to the listbox for each row that is read. The data reader uses the `GetString` method with an argument of 0 to get the first column from the result set row.

```
listEmployees.BeginUpdate();  
while( reader.Read() ) {  
    listEmployees.Items.Add( reader.GetString( 0 ) );  
}
```



```
}  
listEmployees.EndUpdate();
```

For more information about the `SADataReader` object, see [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#).

### Finishing off

The following code at the end of the method closes the data reader and connection objects.

```
reader.Close();  
conn.Close();
```

### Error handling

Any errors that occur during execution and that originate with SQL Anywhere .NET Data Provider objects are handled by displaying them in a window. The following code catches the error and displays its message:

```
catch( SAException ex ) {  
    MessageBox.Show( ex.Errors[0].Message );  
}
```

For more information about the `SAException` object, see [“SAException class \[SQL Anywhere .NET\]” on page 246](#).

## Tutorial: Using the Table Viewer code sample

The `TableViewer` project uses the .NET Data Provider to connect to a database, execute SQL statements, and display the results using a `DataGrid` object.

### Prerequisites

Microsoft Visual Studio

### Context and remarks

The `TableViewer` project is included with the SQL Anywhere samples. The `Table Viewer` project is more complex than the `Simple` project. You can use it to connect to a database, select a table, and execute SQL statements on the database.

### Run the Table Viewer code sample in Visual Studio

1. Start Visual Studio.
2. Click **File » Open » Project**.
3. Browse to `%SQLANYSAAMP12%\SQLAnywhere\ADO.NET\TableViewer` and open the `TableViewer.sln` project.
4. If you want to use the SQL Anywhere .NET Data Provider in a project, you must add a reference to the Data Provider DLL. This has already been done in the `Table Viewer` code sample. To view the

reference to the Data Provider (*iAnywhere.Data.SQLAnywhere*), open the **References** folder in the **Solution Explorer** window.

5. You must also add a `using` directive to your source code to reference the Data Provider classes. This has already been done in the Table Viewer code sample. To view the `using` directive:
  - Open the source code for the project. In the **Solution Explorer** window, right-click *TableViewer.cs* and click **View Code**.
  - In the `using` directives in the top section, you should see the following line:

```
using iAnywhere.Data.SQLAnywhere;
```

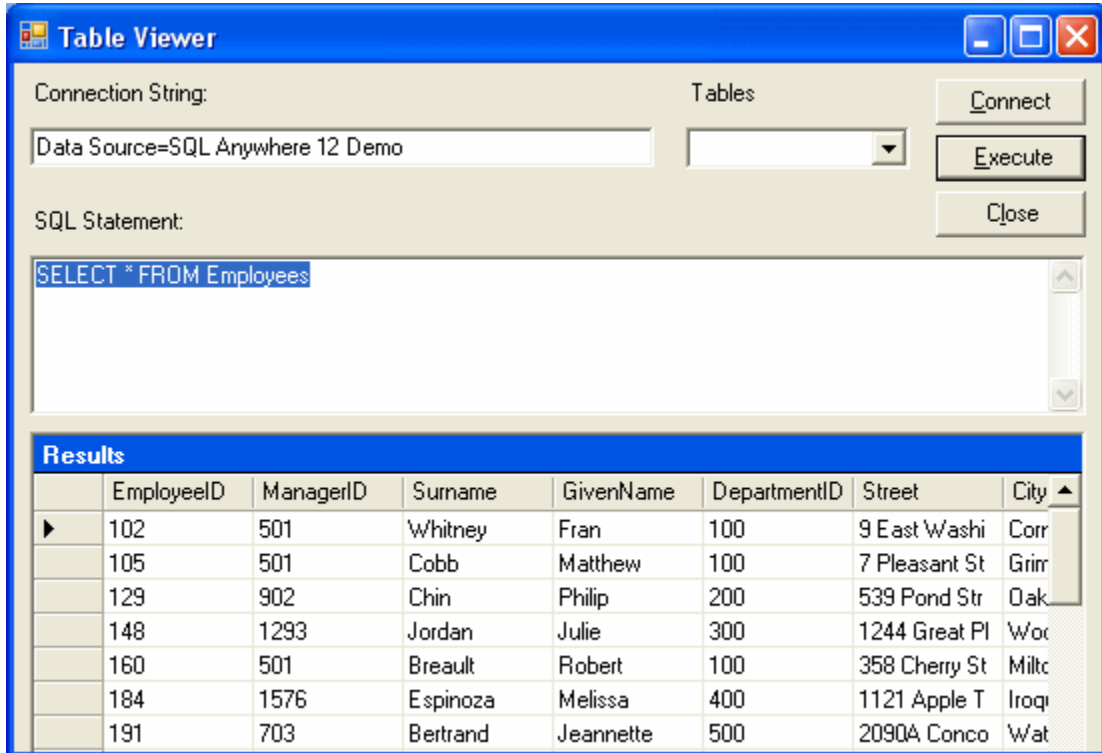
This line is required for C# projects. If you are using Visual Basic, you need to add an `Imports` line to your source code.

6. Click **Debug** » **Start Without Debugging** or press Ctrl+F5 to run the Table Viewer sample.

The application connects to the SQL Anywhere sample database.

7. In the **Table Viewer** window, click **Connect**.
8. In the **Table Viewer** window, click **Execute**.

The application retrieves the data from the Employees table in the sample database and puts the query results in the **Results** datagrid, as follows:



You can also execute other SQL statements from this application: type a SQL statement in the **SQL Statement** pane, and then click **Execute**.

9. Close the **Table Viewer** window to shut down the application and disconnect from the sample database. This also shuts down the database server.

## Results

You have built and executed a .NET application that uses the .NET Data Provider to connect to a database, execute SQL statements, and display the results using a DataGrid object.

## Next

None.

## Example

The complete application can be found in the SQL Anywhere samples directory at `%SQLANY%SAMP12%\SQLAnywhere\ADO.NET\TableViewer`.

## See also

- [“Using the .NET Data Provider in a Visual Studio project”](#) on page 42
- [“Understanding the Table Viewer sample project”](#) on page 81

## Understanding the Table Viewer sample project

This section illustrates some key features of the SQL Anywhere .NET Data Provider by walking through some of the code from the Table Viewer code sample. The Table Viewer project uses the SQL Anywhere sample database, *demo.db*, which is held in your SQL Anywhere samples directory.

For information about the location of the SQL Anywhere samples directory, see [“Samples directory” \[SQL Anywhere Server - Database Administration\]](#).

For information about the sample database, including the tables in the database and the relationships between them, see [“SQL Anywhere sample database” \[SQL Anywhere 12 - Introduction\]](#).

In this section the code is described a few lines at a time. Not all code from the sample is included here. To see all the code, open the sample project in `%SQLANY%SAMP12%\SQLAnywhere\ADO.NET\TableViewer`.

## Declaring controls

The following code declares a couple of Labels named `label1` and `label2`, a TextBox named `txtConnectionString`, a button named `btnConnect`, a TextBox named `txtSQLStatement`, a button named `btnExecute`, and a DataGrid named `dgResults`.

```
private System.Windows.Forms.Label label1;  
private System.Windows.Forms.TextBox txtConnectionString;  
private System.Windows.Forms.Label label2;  
private System.Windows.Forms.Button btnConnect;
```

```
private System.Windows.Forms.TextBox txtSQLStatement;  
private System.Windows.Forms.Button btnExecute;  
private System.Windows.Forms.DataGrid dgResults;
```

### Declaring a connection object

The `SACConnection` type is used to declare an uninitialized SQL Anywhere connection object. The `SACConnection` object is used to represent a unique connection to a SQL Anywhere data source.

```
private SACConnection _conn;
```

For more information about the `SACConnection` class, see [“SACConnection class \[SQL Anywhere .NET\]” on page 160](#).

### Connecting to the database

The `Text` property of the `txtConnectionString` object has a default value of "Data Source=SQL Anywhere 12 Demo". This value can be overridden by the application user by typing a new value into the `txtConnectionString` text box. You can see how this default value is set by opening up the region or section in *TableViewer.cs* labeled Windows Form Designer Generated Code. In this section, you find the following line of code.

```
this.txtConnectionString.Text = "Data Source=SQL Anywhere 12 Demo";
```

Later, the `SACConnection` object uses the connection string to connect to a database. The following code creates a new connection object with the connection string using the `SACConnection` constructor. It then establishes the connection by using the `Open` method.

```
_conn = new SACConnection( txtConnectionString.Text );  
_conn.Open();
```

For more information about the `SACConnection` constructor, see [“SACConnection.SACConnection constructor \[SQL Anywhere .NET\]” on page 162](#).

### Defining a query

The `Text` property of the `txtSQLStatement` object has a default value of "SELECT \* FROM Employees". This value can be overridden by the application user by typing a new value into the `txtSQLStatement` text box.

The SQL statement is executed using an `SACCommand` object. The following code declares and creates a command object using the `SACCommand` constructor. This constructor accepts a string representing the query to be executed, along with the `SACConnection` object that represents the connection that the query is executed on.

```
SACCommand cmd = new SACCommand( txtSQLStatement.Text.Trim(), _conn );
```

For more information about the `SACCommand` object, see [“SACCommand class \[SQL Anywhere .NET\]” on page 122](#).

### Displaying the results

The results of the query are obtained using an `SADataReader` object. The following code declares and creates an `SADataReader` object using the `ExecuteReader` constructor. This constructor is a member of the

SACommand object, cmd, that was declared previously. ExecuteReader sends the command text to the connection for execution and builds an SADATAReader.

```
SADATAReader dr = cmd.ExecuteReader();
```

The following code connects the SADATAReader object to the DataGrid object, which causes the result columns to appear on the screen. The SADATAReader object is then closed.

```
dgResults.DataSource = dr;
dr.Close();
```

For more information about the SADATAReader object, see [“SADATAReader class \[SQL Anywhere .NET\]” on page 214](#).

## Error handling

If there is an error when the application attempts to connect to the database or when it populates the Tables combo box, the following code catches the error and displays its message:

```
try {
    _conn = new SACConnection( txtConnectionString.Text );
    _conn.Open();

    SACommand cmd = new SACommand( "SELECT table_name FROM sys.systable " +
        "WHERE creator = 101 AND table_type != 'TEXT'", _conn );
    SADATAReader dr = cmd.ExecuteReader();

    comboBoxTables.Items.Clear();
    while ( dr.Read() ) {
        comboBoxTables.Items.Add( dr.GetString( 0 ) );
    }
    dr.Close();
} catch( SAException ex ) {
    MessageBox.Show( ex.Errors[0].Source + " : " + ex.Errors[0].Message + " (" +
        ex.Errors[0].NativeError.ToString() + ")",
        "Failed to connect" );
}
```

For more information about the SAException object, see [“SAException class \[SQL Anywhere .NET\]” on page 246](#).

## Tutorial: Developing a simple .NET database application with Visual Studio

This section contains a tutorial that takes you through the steps of building the Simple Viewer .NET database application using Visual Studio.

## Lesson 1: Creating a table viewer

This tutorial is based on Visual Studio and the .NET Framework. The complete application can be found in the ADO.NET project `%SQLANYSAMPI2%\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln`.

In this tutorial, you use Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider to create an application that accesses one of the tables in the SQL Anywhere sample database, allowing you to examine rows and perform updates.

### Develop a database application with Visual Studio

1. Start Visual Studio.
2. Click **File** » **New** » **Project**.

The **New Project** window appears.

- a. In the left pane of the **New Project** window, click either **Visual Basic** or **Visual C#** for the programming language.
  - b. From the **Windows** subcategory, click **Windows Application** (VS 2005) or **Windows Forms Application** (VS 2008).
  - c. In the project **Name** field, type **MySimpleViewer**.
  - d. Click **OK** to create the new project.
3. Click **View** » **Server Explorer**.
  4. In the **Server Explorer** window, right-click **Data Connections** and click **Add Connection**.

A new connection named **SQL Anywhere.demo12** appears in the **Server Explorer** window.

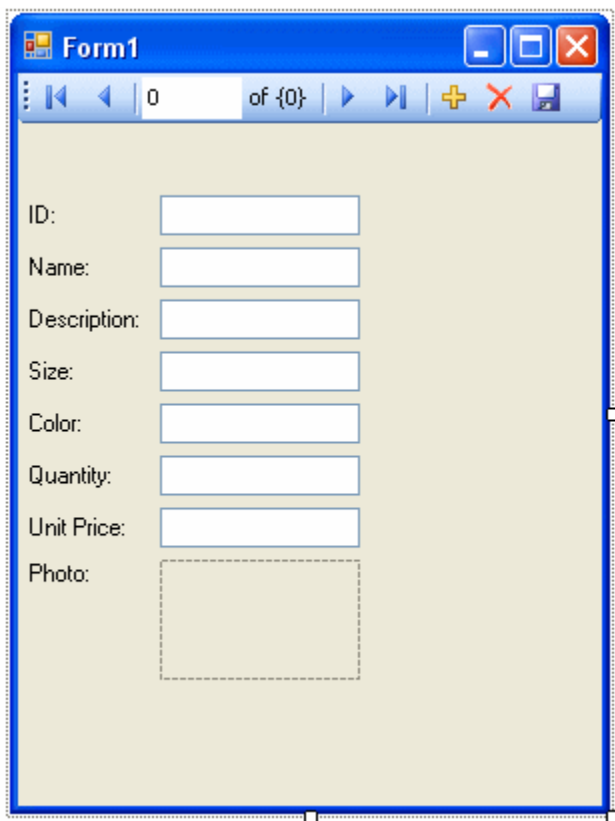
5. In the **Add Connection** window:
  - a. If you have never used **Add Connection** for other projects, then you see a list of data sources. Click **SQL Anywhere** from the list of data sources presented.  
If you have used **Add Connection** before, then click **Change** to change the data source to **SQL Anywhere**.
  - b. Under **Data Source**, click **ODBC Data Source Name** and type **SQL Anywhere 12 Demo**.

#### Note

When using the Visual Studio Add Connection wizard on 64-bit Windows, only the 64-bit System Data Source Names (DSN) are included with the User Data Source Names. Any 32-bit System Data Source Names are not displayed. In Visual Studio's 32-bit design environment, the Test Connection button will attempt to establish a connection using the 32-bit equivalent of the 64-bit System DSN. If the 32-bit System DSN does not exist, the test will fail.

- c. Click **Test Connection** to verify that you can connect to the sample database.
- d. Click **OK**.

6. Expand the **SQL Anywhere.demo12** connection in the **Server Explorer** window until you see the table names. If you are using Visual Studio 2005, then try the following:
  - a. Right-click the Products table and click **Show Table Data**.  
This shows the rows and columns of the Products table in a window.
  - b. Close the table data window.
7. Click **Data » Add New Data Source**.
8. In the **Data Source Configuration Wizard**, do the following:
  - a. On the **Data Source Type** page, click **Database**, then click **Next**.
  - b. On the **Data Connection** page, click **SQL Anywhere.demo12**, then click **Next**.
  - c. On the **Save The Connection String** page, make sure that **Yes, Save The Connection As is** chosen and click **Next**.
  - d. On the **Choose Your Database Objects** page, click **Tables**, then click **Finish**.
9. Click **Data » Show Data Sources**.  
The **Data Sources** window appears.  
Expand the Products table in the **Data Sources** window.
  - a. Click Products, then click **Details** from the dropdown list.
  - b. Click Photo, then click **Picture Box** from the dropdown list.
  - c. Click Products and drag it to your form (Form1).



A dataset control and several labeled text fields appear on the form.

10. On the form, click the picture box next to Photo.
  - a. Change the shape of the box to a square.
  - b. Click the right-arrow in the upper-right corner of the picture box.  
The **Picture Box Tasks** window opens.
  - c. From the **Size Mode** dropdown list, click **Zoom**.
  - d. To close the **Picture Box Tasks** window, click anywhere outside the window.
11. Build and run the project.
  - a. Click **Build » Build Solution**.
  - b. Click **Debug » Start Debugging**.  
The application connects to the SQL Anywhere sample database and displays the first row of the Products table in the text boxes and picture box.





- c. You can use the buttons on the control to scroll through the rows of the result set.
- d. You can go directly to a row in the result set by entering the row number in the scroll control.
- e. You can update values in the result set using the text boxes and save them by clicking the **Save Data** button.

You have now created a simple, yet powerful, .NET application using Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider.

12. Shut down the application and then save your project.

## Lesson 2: Adding a synchronizing data control

This tutorial is a continuation of the tutorial described in “[Lesson 1: Creating a table viewer](#)” on page 84.

The complete application can be found in the ADO.NET project *%SQLANYWHERE%\ADO.NET\SimpleViewer\SimpleViewer.sln*.

In this tutorial, you add a datagrid control to the form developed in the previous tutorial. This control updates automatically as you navigate through the result set.

### Add a datagrid control

1. Start Visual Studio and load your MySimpleViewer project. This project was created in [“Lesson 1: Creating a table viewer”](#) on page 84.
2. Right-click DataSet1 in the **Data Sources** window and click **Edit DataSet With Designer**.
3. Right-click an empty area in the **DataSet Designer** window and click **Add » TableAdapter**.
4. In the **TableAdapter Configuration Wizard**:
  - a. On the **Choose Your Data Connection** page, click **Next**.
  - b. On the **Choose A Command Type** page, click **Use SQL Statements**, then click **Next**.
  - c. On the **Enter A SQL Statement** page, click **Query Builder**.
  - d. On the **Add Table** window, click the **Views** tab, then click **ViewSalesOrders**, and then click **Add**.
  - e. Click **Close** to close the **Add Table** window.
5. Expand the **Query Builder** window so that all sections of the window are visible.
  - a. Expand the **ViewSalesOrders** window so that all the checkboxes are visible.
  - b. Click **Region**.
  - c. Click **Quantity**.
  - d. Click **ProductID**.
  - e. In the grid below the **ViewSalesOrders** window, clear the checkbox under **Output** for the ProductID column.
  - f. For the ProductID column, type a question mark (?) in the **Filter** cell. This generates a WHERE clause for ProductID.

A SQL query has been built that looks like the following:

```
SELECT    Region, Quantity
FROM      GROUPO.ViewSalesOrders
WHERE     (ProductID = :Param1)
```

6. Modify the SQL query as follows:
  - a. Change Quantity to SUM(Quantity) AS TotalSales.
  - b. Add GROUP BY Region to the end of the query following the WHERE clause.

The modified SQL query now looks like this:

```
SELECT    Region, SUM(Quantity) as TotalSales
FROM      GROUPO.ViewSalesOrders
WHERE     (ProductID = :Param1)
GROUP BY Region
```

7. Click **OK**.

8. Click **Finish**.

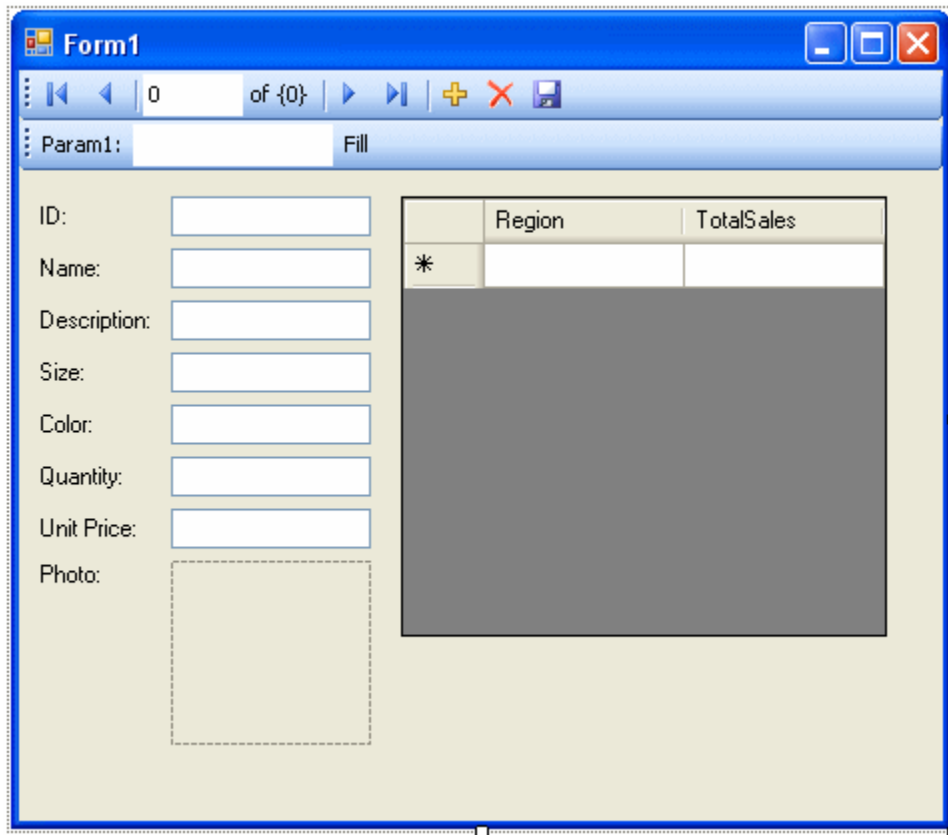
A new **TableAdapter** called **ViewSalesOrders** has been added to the **DataSet Designer** window.

9. Click the form design tab (Form1).

- Stretch the form to the right to make room for a new control.

10. Expand ViewSalesOrders in the **Data Sources** window.

- a. Click ViewSalesOrders and click **DataGridView** from the dropdown list.
- b. Click ViewSalesOrders and drag it to your form (Form1).

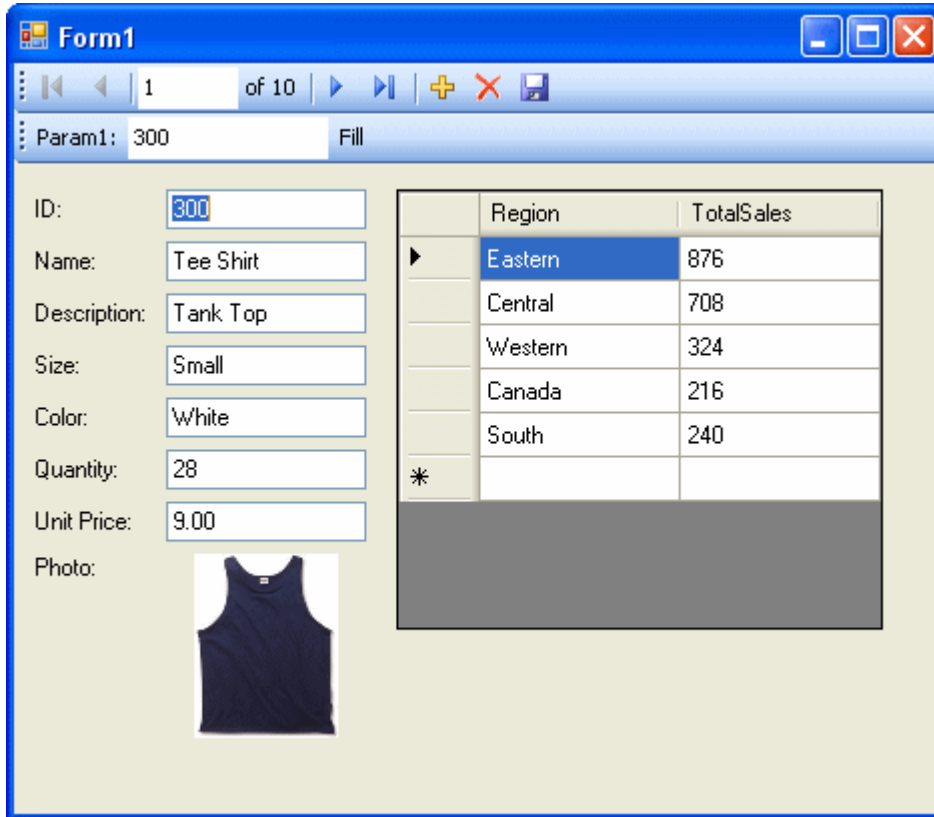


A datagrid view control appears on the form.

11. Build and run the project.

- Click **Build » Build Solution**.
- Click **Debug » Start Debugging**.
- In the **Param1** text box, enter a product ID number such as 300 and click **Fill**.

The datagrid view displays a summary of sales by region for the product ID entered.



You can also use the other control on the form to move through the rows of the result set.

It would be ideal, however, if both controls could stay synchronized with each other. The next few steps show how to do this.

12. Shut down the application and then save your project.

13. Delete the Fill strip on the form since you do not need it.

- On the design form (Form1), right-click the Fill strip to the right of the word **Fill**, then click **Delete**.

The Fill strip is removed from the form.

14. Synchronize the two controls as follows.

- a. On the design form (Form1), right-click the ID text box, then click **Properties**.
- b. Click the **Events** button (it appears as a lightning bolt).
- c. Scroll down until you find the **TextChanged** event.
- d. Click **TextChanged**, then click **FillToolStripButton\_Click** from the dropdown list. If you are using Visual Basic, the event is called **FillToolStripButton\_Click**.
- e. Double-click **FillToolStripButton\_Click** and the form's code window opens on the `fillToolStripButton_Click` event handler.

- f. Find the reference to `param1ToolStripTextBox` and change this to `iDTextBox`. If you are using Visual Basic, the text box is called `IDTextBox`.
  - g. Rebuild and run the project.
15. The application form now appears with a single navigation control.
- The datagrid view displays an updated summary of sales by region corresponding to the current product as you move through the result set.

	Region	TotalSales
▶	Eastern	1130
	Central	1116
	Western	360
	Canada	252
	South	420
*		

You have now added a control that updates automatically as you navigate through the result set.

16. Shut down the application and then save your project.

In these tutorials, you saw how the powerful combination of Microsoft Visual Studio, the Server Explorer, and the SQL Anywhere .NET Data Provider can be used to create database applications.

## SQL Anywhere ASP.NET Providers

The SQL Anywhere ASP.NET providers replace the standard ASP.NET providers for SQL Server, and allow you to run your website on a SQL Anywhere database. There are five providers:

- **Membership Provider** The membership provider provides authentication and authorization services. Use the membership provider to create new users and passwords, and validate the identity of users.
- **Roles Provider** The roles provider provides methods for creating roles, adding users to roles, and deleting roles. Use the roles provider to assign users to groups and manage permissions.
- **Profiles Provider** The profiles provider provides methods for reading, storing, and retrieving user information. Use the profiles provider to save user preferences.
- **Web Parts Personalization Provider** The web parts personalization provider provides methods for loading and storing the personalized content and layout of web pages. Use the web parts personalization provider to allow users to create personalized views of your website.
- **Health Monitoring Provider** The health monitoring provider provides methods for monitoring the status of deployed web applications. Use the health monitoring provider to monitor application performance, identify failing applications or systems, and log and review significant events.

The SQL Anywhere database server schema used by the SQL Anywhere ASP.NET providers is identical to the schema used by the standard ASP.NET providers. The methodology used to manipulate and store data are identical.

When you have finished setting up the SQL Anywhere ASP.NET providers, you can use the Visual Studio ASP.NET Web Site Administration Tool to create and manage users and roles. You can also use the Visual Studio Login, LoginView, and PasswordRecovery tools to add security your web site. Use the static wrapper classes to access more advanced provider functions, or to make your own login controls.

**Note**

A white paper called *Tutorial: Creating an ASP.NET Web Page Using SQL Anywhere* is available to demonstrate how to use SQL Anywhere and Visual Studio 2010 to build a database-driven ASP.NET web site. See [www.sybase.com/detail?id=1080238](http://www.sybase.com/detail?id=1080238).

## Adding the SQL Anywhere ASP.NET provider schema to the database

To implement the SQL Anywhere ASP.NET providers you can create a new database, or add the schema to an existing database.

To add the schema to an existing SQL Anywhere database, run *SASetupAspNet.exe*. When executed, *SASetupAspNet.exe* connects to an existing SQL Anywhere database and creates tables and stored procedures required by the SQL Anywhere ASP.NET providers. All SQL Anywhere ASP.NET provider resources are prefixed with *aspnet\_*. To minimize naming conflicts with existing database resources you can install provider database resources under any database user.

You can use a wizard or the command line to run *SASetupAspNet.exe*. To access the wizard, run the application, or execute a command line statement without arguments. When using the command line to

access the *SASetupAspNet.exe*, use the question mark (-?) argument to display detailed help for configuring the database.

### Setting up the database connection

It is recommended that you specify a connection string for a user with DBA authority. A user with DBA authority can create resources for other users that might not have the necessary permissions. Alternatively, specify a connection string for a user with RESOURCE authority. The RESOURCE authority allows a user to create database objects, such as tables, views, stored procedures, and triggers. The RESOURCE authority is not inherited through group membership, and can be granted only by a user with DBA authority.

### Specifying a resource owner

The wizard and command line allow you to specify the owner of the new resources. By default, the owner of new resources is DBA. When you specify the connection string for the SQL Anywhere ASP.NET providers, specify the user as DBA. You do not need to grant the user any permissions; the DBA owns the resources and has full permissions on the tables and stored procedures.

### Selecting features and preserving data

You can add or remove specific features. Common components are installed automatically. Selecting **Remove** for an uninstalled feature has no effect; selecting **Add** for a feature already installed reinstalls the feature. By default, the data in tables associated with the selected feature is preserved. If a user significantly changes the schema of a table, it might not be possible to automatically preserve the data stored in it. If a clean reinstall is required, data preservation can be turned off.

It is recommended that the membership and roles providers are installed together. The effectiveness of the Visual Studio ASP.NET Web Site Administration Tool is reduced when the membership provider is not installed with the roles provider.

### Further reading

A white paper called *Tutorial: Creating an ASP.NET Web Page Using SQL Anywhere* is available to demonstrate how to use SQL Anywhere and Visual Studio 2010 to build a database-driven ASP.NET web site. See [www.sybase.com/detail?id=1080238](http://www.sybase.com/detail?id=1080238).

## Connection string registration

There are two methods for registering the connection string:

- You can register an ODBC data source in the ODBC Data Source Administrator, and reference it by name.
- You can specify a full SQL Anywhere connection string. For example:

```
connectionString="SERVER=MyServer;DBN=MyDatabase;UID=DBA;PWD=sql"
```

When you add the <connectionStrings> element to the *web.config* file, the connection string and its provider can be referenced by the application. Updates can be implemented in a single location.

### XML code sample for connection string registration

```
<connectionStrings>
  <add name="MyConnectionString"
        connectionString="DSN=MyDataSource"
        providerName="iAnywhere.Data.SQLAnywhere" />
</connectionStrings>
```

## Registration of SQL Anywhere ASP.NET providers

Your web application must be configured to use the SQL Anywhere ASP.NET providers and not the default providers. To register SQL Anywhere ASP.NET providers:

- Add a reference to the `iAnywhere.Web.Security` assembly to your web site.
- Add an entry for each provider to the `<system.web>` element in `web.config` file.
- Add the name of the SQL Anywhere ASP.NET provider to the `defaultProvider` attribute in the application.

The provider database can store data for multiple applications. For each application, the `applicationName` attribute must be the same for each SQL Anywhere ASP.NET provider. If you do not specify an `applicationName` value, an identical name is assigned to each provider in the provider database.

To reference a previously registered connection string, replace the `connectionString` attribute with the `connectionStringName` attribute.

### XML code sample for Membership Provider registration

```
<membership defaultProvider="SAMembershipProvider">
  <providers>
    <add name="SAMembershipProvider"
          type="iAnywhere.Web.Security.SAMembershipProvider"
          connectionStringName="MyConnectionString"
          applicationName="MyApplication"
          commandTimeout="30"
          enablePasswordReset="true"
          enablePasswordRetrieval="false"
          maxInvalidPasswordAttempts="5"
          minRequiredNonalphanumericCharacters="1"
          minRequiredPasswordLength="7"
          passwordAttemptWindow="10"
          passwordFormat="Hashed"
          requiresQuestionAndAnswer="true"
          requiresUniqueEmail="true"
          passwordStrengthRegularExpression="" />
  </providers>
</membership>
```

For column descriptions, see [“Membership provider XML attributes” on page 96](#).

### XML code sample for Roles Provider registration

```
<roleManager enabled="true" defaultProvider="SARoleProvider">
  <providers>
```



```

    <add name="SARoleProvider"
        type="iAnywhere.Web.Security.SARoleProvider"
        connectionStringName="MyConnectionString"
        applicationName="MyApplication"
        commandTimeout="30" />
  </providers>
</roleManager>

```

For column descriptions, see [“Roles provider table schema” on page 97](#).

### XML code sample for Profiles Provider registration

```

<profile defaultProvider="SAProfileProvider">
  <providers>
    <add name="SAProfileProvider"
        type="iAnywhere.Web.Security.SAProfileProvider"
        connectionStringName="MyConnectionString"
        applicationName="MyApplication"
        commandTimeout="30" />
  </providers>
  <properties>
    <add name="UserString" type="string"
        serializeAs="Xml" />
    <add name="UserObject" type="object"
        serializeAs="Binary" />
  </properties>
</profile>

```

For column descriptions, see [“Profile provider table schema” on page 97](#).

### XML code sample for Personalization Provider registration

```

<webParts>
  <personalization defaultProvider="SAPersonalizationProvider">
    <providers>
      <add name="SAPersonalizationProvider"
          type="iAnywhere.Web.Security.SAPersonalizationProvider"
          connectionStringName="MyConnectionString"
          applicationName="MyApplication"
          commandTimeout="30" />
    </providers>
  </personalization>
</webParts>

```

For column descriptions, see [“Web Part Personalization provider table schema” on page 98](#).

### XML code sample for Health Monitoring Provider registration

For more information about setting up health monitoring, see the Microsoft web page [How To: Use Health Monitoring in ASP.NET 2.0 \(http://msdn.microsoft.com/en-us/library/ms998306.aspx\)](http://msdn.microsoft.com/en-us/library/ms998306.aspx).

```

<healthMonitoring enabled="true">
  ...
  <providers>
    <add name="SAWebEventProvider"
        type="iAnywhere.Web.Security.SAWebEventProvider"
        connectionStringName="MyConnectionString"
        commandTimeout="30"
        bufferMode="Notification"
        maxEventDetailsLength="Infinite" /
  </providers>

```

```
</healthMonitoring>
```

For column descriptions, see [“Health Monitoring provider table schema” on page 98.](#)

## Membership provider XML attributes

Column name	Description
name	The name of the provider.
type	<code>iAnywhere.Web.Security.SAMembershipProvider</code>
connectionStringName	The name of a connection string specified in the <code>&lt;connectionStrings&gt;</code> element.
connectionString	The connection string. Optional. Required if <code>connectionStringName</code> is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.
enablePasswordReset	Valid entries are true or false.
enablePasswordRetrieval	Valid entries are true or false.
maxInvalidPasswordAttempts	Valid entries are true or false.
minRequiredNonalphanumericCharacters	The minimum number of special characters that must be present in a valid password.
minRequiredPasswordLength	The minimum length required for a password.
passwordAttemptWindow	The time window between which consecutive failed attempts to provide a valid password or password answer are tracked.
passwordFormat	Valid entries are Clear, Hashed, or Encrypted.
requiresQuestionAndAnswer	Valid entries are true or false.
requiresUniqueEmail	Valid entries are true or false.
passwordStrengthRegularExpression	The regular expression used to evaluate a password.

## Roles provider table schema

SARoleProvider stores role information in the aspnet\_Roles table of the provider database. The namespace associated with SARoleProvider is iAnywhere.Web.Security. Each record in the Roles table corresponds to one role.

SARoleProvider uses the aspnet\_UsersInRoles table to map roles to users.

Column name	Description
name	The name of the provider.
type	iAnywhere.Web.Security.SARoleProvider
connectionStringName	The name of a connection string specified in the <connectionStrings> element.
connectionString	The connection string. Optional. Required if connectionStringName is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

## Profile provider table schema

SAProfileProvider stores profile data in the aspnet\_Profile table of the provider database. The namespace associated with SAProfileProvider is iAnywhere.Web.Security. Each record in the Profile table corresponds to one user's persisted profile properties.

Column name	Description
name	The name of the provider.
type	iAnywhere.Web.Security.SAProfileProvider
connectionStringName	The name of a connection string specified in the <connectionStrings> element.
connectionString	The connection string. Optional. Required if connectionStringName is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

## Web Part Personalization provider table schema

SAPersonalizationProvider preserves personalized user content in the aspnet\_Paths table of the provider database. The namespace associated with SAPersonalizationProvider is iAnywhere.Web.Security.

SAPersonalizationProvider uses the aspnet\_PersonalizationPerUser and aspnet\_PersonalizationAllUsers tables to define the path for which the Web Parts personalization state has been saved. The PathID columns point to the column of the same name in the aspnet\_Paths table.

Column name	Description
name	The name of the provider.
type	iAnywhere.Web.Security.SAPersonalizationProvider
connectionStringName	The name of a connection string specified in the <connectionStrings> element.
connectionString	The connection string. Optional. Required if connectionStringName is not specified.
applicationName	The application name with which to associate provider data.
commandTimeout	The timeout value, in seconds, for server calls.

## Health Monitoring provider table schema

SAWebEventProvider logs web events in the aspnet\_WebEvent\_Events table of the provider database. The namespace associated with SAWebEventProvider is iAnywhere.Web.Security. Each record in the WebEvents\_Events table corresponds to one web event.

For more information about setting up health monitoring, see the Microsoft web page How To: Use Health Monitoring in ASP.NET 2.0 (<http://msdn.microsoft.com/en-us/library/ms998306.aspx>).

Column name	Description
name	The name of the provider.
type	iAnywhere.Web.Security.SAWebEventProvider
connectionStringName	The name of a connection string specified in the <connectionStrings> element.
connectionString	The connection string. Optional. Required if connectionStringName is not specified.

Column name	Description
commandTimeout	The timeout value, in seconds, for server calls.
maxEventDetailsLength	The maximum length of the details string for each event or Infinite

## SQL Anywhere .NET API reference

### Namespace

`iAnywhere.Data.SQLAnywhere`

## SABulkCopy class

Efficiently bulk load a SQL Anywhere table with data from another source.

### Visual Basic syntax

```
Public NotInheritable Class SABulkCopy Implements System.IDisposable
```

### C# syntax

```
public sealed class SABulkCopy : System.IDisposable
```

### Base classes

- [System.IDisposable](#)

### Members

All members of SABulkCopy class, including all inherited members.

Name	Description
<a href="#">SABulkCopy constructor</a>	Initializes an SABulkCopy object.
<a href="#">Close method</a>	Closes the SABulkCopy instance.
<a href="#">Dispose method</a>	Disposes of the SABulkCopy instance.
<a href="#">WriteToServer method</a>	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.
<a href="#">BatchSize property</a>	Gets or sets the number of rows in each batch.

Name	Description
<a href="#">BulkCopyTimeout property</a>	Gets or sets the number of seconds for the operation to complete before it times out.
<a href="#">ColumnMappings property</a>	Returns a collection of SABulkCopyColumnMapping items.
<a href="#">DestinationTableName property</a>	Gets or sets the name of the destination table on the server.
<a href="#">NotifyAfter property</a>	Gets or sets the number of rows to be processed before generating a notification event.
<a href="#">SARowsCopied event</a>	This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

**Remarks**

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

**Implements:** System.IDisposable

## SABulkCopy constructor

Initializes an SABulkCopy object.

**Overload list**

Name	Description
<a href="#">SABulkCopy(SAConnection) constructor</a>	Initializes an SABulkCopy object.
<a href="#">SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction) constructor</a>	Initializes an SABulkCopy object.
<a href="#">SABulkCopy(string) constructor</a>	Initializes an SABulkCopy object.
<a href="#">SABulkCopy(string, SABulkCopyOptions) constructor</a>	Initializes an SABulkCopy object.

## SABulkCopy(SAConnection) constructor

Initializes an SABulkCopy object.

**Visual Basic syntax**

```
Public Sub New(ByVal connection As SAConnection)
```

**C# syntax**

```
public SABulkCopy(SAConnection connection)
```

**Parameters**

- **connection** The already open SAConnection that will be used to perform the bulk-copy operation. If the connection is not open, an exception is thrown in WriteToServer.

**Remarks**

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

**SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction) constructor**

Initializes an SABulkCopy object.

**Visual Basic syntax**

```
Public Sub New(  
    ByVal connection As SAConnection,  
    ByVal copyOptions As SABulkCopyOptions,  
    ByVal externalTransaction As SATransaction  
)
```

**C# syntax**

```
public SABulkCopy(  
    SAConnection connection,  
    SABulkCopyOptions copyOptions,  
    SATransaction externalTransaction  
)
```

**Parameters**

- **connection** The already open SAConnection that will be used to perform the bulk-copy operation. If the connection is not open, an exception is thrown in WriteToServer.
- **copyOptions** A combination of values from the SABulkCopyOptions enumeration that determines which data source rows are copied to the destination table.
- **externalTransaction** An existing SATransaction instance under which the bulk copy will occur. If externalTransaction is not NULL, then the bulk-copy operation is done within it. It is an error to specify both an external transaction and the UseInternalTransaction option.

**Remarks**

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

**SABulkCopy(string) constructor**

Initializes an SABulkCopy object.

### Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

### C# syntax

```
public SABulkCopy(string connectionString)
```

### Parameters

- **connectionString** The string defining the connection that will be opened for use by the SABulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.

### Remarks

This syntax opens a connection during WriteToServer using *connectionString*. The connection is closed at the end of WriteToServer.

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

## SABulkCopy(string, SABulkCopyOptions) constructor

Initializes an SABulkCopy object.

### Visual Basic syntax

```
Public Sub New(  
    ByVal connectionString As String,  
    ByVal copyOptions As SABulkCopyOptions  
)
```

### C# syntax

```
public SABulkCopy(  
    string connectionString,  
    SABulkCopyOptions copyOptions  
)
```

### Parameters

- **connectionString** The string defining the connection that will be opened for use by the SABulkCopy instance. A connection string is a semicolon-separated list of keyword=value pairs.
- **copyOptions** A combination of values from the SABulkCopyOptions enumeration that determines which data source rows are copied to the destination table.

### Remarks

This syntax opens a connection during WriteToServer using *connectionString*. The connection is closed at the end of WriteToServer. The *copyOptions* parameter has the effects described above.

The SABulkCopy class is not available in the .NET Compact Framework 2.0.



## Close method

Closes the SABulkCopy instance.

### Visual Basic syntax

```
Public Sub Close()
```

### C# syntax

```
public void Close()
```

## Dispose method

Disposes of the SABulkCopy instance.

### Visual Basic syntax

```
Public Sub Dispose()
```

### C# syntax

```
public void Dispose()
```

## WriteToServer method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.

### Overload list

Name	Description
<a href="#">WriteToServer(DataRow[]) method</a>	Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.
<a href="#">WriteToServer(DataTable) method</a>	Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName property of the SABulkCopy object.
<a href="#">WriteToServer(DataTable, DataRowState) method</a>	Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName property of the SABulkCopy object.
<a href="#">WriteToServer(IDataReader) method</a>	Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName property of the SABulkCopy object.

## WriteToServer(DataRow[]) method

Copies all rows in the supplied array of System.Data.DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.

### Visual Basic syntax

```
Public Sub WriteToServer(ByVal rows As DataRow())
```

### C# syntax

```
public void WriteToServer(DataRow[] rows)
```

### Parameters

- **rows** An array of System.Data.DataRow objects that will be copied to the destination table.

### Remarks

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

### See also

- [“SABulkCopy.DestinationTableName property \[SQL Anywhere .NET\]” on page 107](#)

## WriteToServer(DataTable) method

Copies all rows in the supplied System.Data.DataTable to a destination table specified by the DestinationTableName property of the SABulkCopy object.

### Visual Basic syntax

```
Public Sub WriteToServer(ByVal table As DataTable)
```

### C# syntax

```
public void WriteToServer(DataTable table)
```

### Parameters

- **table** A System.Data.DataTable whose rows will be copied to the destination table.

### Remarks

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

### See also

- [“SABulkCopy.DestinationTableName property \[SQL Anywhere .NET\]” on page 107](#)

## WriteToServer(DataTable, DataRowState) method

Copies all rows in the supplied System.Data.DataTable with the specified row state to a destination table specified by the DestinationTableName property of the SABulkCopy object.

**Visual Basic syntax**

```
Public Sub WriteToServer(  
    ByVal table As DataTable,  
    ByVal rowState As DataRowState  
)
```

**C# syntax**

```
public void WriteToServer(DataTable table, DataRowState rowState)
```

**Parameters**

- **table** A System.Data.DataTable whose rows will be copied to the destination table.
- **rowState** A value from the System.Data.DataRowState enumeration. Only rows matching the row state are copied to the destination.

**Remarks**

Only those rows matching the row state are copied.

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

**See also**

- [“SABulkCopy.DestinationTableName property \[SQL Anywhere .NET\]” on page 107](#)

**WriteToServer(IDataReader) method**

Copies all rows in the supplied System.Data.IDataReader to a destination table specified by the DestinationTableName property of the SABulkCopy object.

**Visual Basic syntax**

```
Public Sub WriteToServer(ByVal reader As IDataReader)
```

**C# syntax**

```
public void WriteToServer(IDataReader reader)
```

**Parameters**

- **reader** A System.Data.IDataReader whose rows will be copied to the destination table.

**Remarks**

The SABulkCopy class is not available in the .NET Compact Framework 2.0.

**See also**

- [“SABulkCopy.DestinationTableName property \[SQL Anywhere .NET\]” on page 107](#)

## BatchSize property

Gets or sets the number of rows in each batch.

### Visual Basic syntax

```
Public Property BatchSize As Integer
```

### C# syntax

```
public int BatchSize {get;set;}
```

### Remarks

At the end of each batch, the rows in the batch are sent to the server.

The number of rows in each batch. The default is 0.

Setting this property to zero causes all the rows to be sent in one batch.

Setting this property to a value less than zero is an error.

If this value is changed while a batch is in progress, the current batch completes and any further batches use the new value.

## BulkCopyTimeout property

Gets or sets the number of seconds for the operation to complete before it times out.

### Visual Basic syntax

```
Public Property BulkCopyTimeout As Integer
```

### C# syntax

```
public int BulkCopyTimeout {get;set;}
```

### Remarks

The default value is 30 seconds.

A value of zero indicates no limit. This should be avoided because it may cause an indefinite wait.

If the operation times out, then all rows in the current transaction are rolled back and an SAException is raised.

Setting this property to a value less than zero is an error.

## ColumnMappings property

Returns a collection of SABulkCopyColumnMapping items.

**Visual Basic syntax**

```
Public ReadOnly Property ColumnMappings As  
SABulkCopyColumnMappingCollection
```

**C# syntax**

```
public SABulkCopyColumnMappingCollection ColumnMappings {get;}
```

**Remarks**

Column mappings define the relationships between columns in the data source and columns in the destination.

By default, it is an empty collection.

The property cannot be modified while WriteToServer is executing.

If ColumnMappings is empty when WriteToServer is executed, then the first column in the source is mapped to the first column in the destination, the second to the second, and so on. This takes place as long as the column types are convertible, there are at least as many destination columns as source columns, and any extra destination columns are nullable.

## DestinationTableName property

Gets or sets the name of the destination table on the server.

**Visual Basic syntax**

```
Public Property DestinationTableName As String
```

**C# syntax**

```
public string DestinationTableName {get;set;}
```

**Remarks**

The default value is a null reference. In Visual Basic it is Nothing.

If the value is changed while WriteToServer is executing, the change has no effect.

If the value has not been set before a call to WriteToServer, an InvalidOperationException is raised.

It is an error to set the value to NULL or the empty string.

## NotifyAfter property

Gets or sets the number of rows to be processed before generating a notification event.

**Visual Basic syntax**

```
Public Property NotifyAfter As Integer
```

### C# syntax

```
public int NotifyAfter {get;set;}
```

### Remarks

Zero is returned if the property has not been set.

Changes made to NotifyAfter, while executing WriteToServer, do not take effect until after the next notification.

Setting this property to a value less than zero is an error.

The values of NotifyAfter and BulkCopyTimeout are mutually exclusive, so the event can fire even if no rows have been sent to the database or committed.

### See also

- [“SABulkCopy.BulkCopyTimeout property \[SQL Anywhere .NET\]” on page 106](#)

## SARowsCopied event

This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

### Visual Basic syntax

```
Public Event SARowsCopied As SARowsCopiedEventHandler
```

### C# syntax

```
public event SARowsCopiedEventHandler SARowsCopied;
```

### Remarks

The receipt of an SARowsCopied event does not imply that any rows have been sent to the database server or committed. You cannot call the Close method from this event.

### See also

- [“SABulkCopy.NotifyAfter property \[SQL Anywhere .NET\]” on page 107](#)

## SABulkCopyColumnMapping class

Defines the mapping between a column in an SABulkCopy instance's data source and a column in the instance's destination table.

### Visual Basic syntax

```
Public NotInheritable Class SABulkCopyColumnMapping
```

**C# syntax**

```
public sealed class SABulkCopyColumnMapping
```

**Members**

All members of SABulkCopyColumnMapping class, including all inherited members.

Name	Description
<a href="#">SABulkCopyColumnMapping constructor</a>	Creates a new column mapping, using column ordinals or names to refer to source and destination columns.
<a href="#">DestinationColumn property</a>	Gets or sets the name of the column in the destination database table being mapped to.
<a href="#">DestinationOrdinal property</a>	Gets or sets the ordinal value of the column in the destination table being mapped to.
<a href="#">SourceColumn property</a>	Gets or sets the name of the column being mapped in the data source.
<a href="#">SourceOrdinal property</a>	Gets or sets ordinal position of the source column within the data source.

**Remarks**

The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

**SABulkCopyColumnMapping constructor**

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

**Overload list**

Name	Description
<a href="#">SABulkCopyColumnMapping() constructor</a>	Creates a new column mapping, using column ordinals or names to refer to source and destination columns.
<a href="#">SABulkCopyColumnMapping(int, int) constructor</a>	Creates a new column mapping, using column ordinals to refer to source and destination columns.
<a href="#">SABulkCopyColumnMapping(int, string) constructor</a>	Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.
<a href="#">SABulkCopyColumnMapping(string, int) constructor</a>	Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination column.

Name	Description
<a href="#">SABulkCopyColumnMapping(string, string) constructor</a>	Creates a new column mapping, using column names to refer to source and destination columns.

## SABulkCopyColumnMapping() constructor

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

### Visual Basic syntax

```
Public Sub New()
```

### C# syntax

```
public SABulkCopyColumnMapping()
```

### Remarks

The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

## SABulkCopyColumnMapping(int, int) constructor

Creates a new column mapping, using column ordinals to refer to source and destination columns.

### Visual Basic syntax

```
Public Sub New(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumnOrdinal As Integer  
)
```

### C# syntax

```
public SABulkCopyColumnMapping(  
    int sourceColumnOrdinal,  
    int destinationColumnOrdinal  
)
```

### Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

### Remarks

The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.



## SABulkCopyColumnMapping(int, string) constructor

Creates a new column mapping, using a column ordinal to refer to the source column and a column name to refer to the destination column.

### Visual Basic syntax

```
Public Sub New(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumn As String  
)
```

### C# syntax

```
public SABulkCopyColumnMapping(  
    int sourceColumnOrdinal,  
    string destinationColumn  
)
```

### Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source. The first column in a data source has ordinal position zero.
- **destinationColumn** The name of the destination column within the destination table.

### Remarks

The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

## SABulkCopyColumnMapping(string, int) constructor

Creates a new column mapping, using a column name to refer to the source column and a column ordinal to refer to the destination column.

### Visual Basic syntax

```
Public Sub New(  
    ByVal sourceColumn As String,  
    ByVal destinationColumnOrdinal As Integer  
)
```

### C# syntax

```
public SABulkCopyColumnMapping(  
    string sourceColumn,  
    int destinationColumnOrdinal  
)
```

### Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table. The first column in a table has ordinal position zero.

## Remarks

The `SABulkCopyColumnMapping` class is not available in the .NET Compact Framework 2.0.

## **SABulkCopyColumnMapping(string, string) constructor**

Creates a new column mapping, using column names to refer to source and destination columns.

### Visual Basic syntax

```
Public Sub New(  
    ByVal sourceColumn As String,  
    ByVal destinationColumn As String  
)
```

### C# syntax

```
public SABulkCopyColumnMapping(  
    string sourceColumn,  
    string destinationColumn  
)
```

### Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumn** The name of the destination column within the destination table.

## Remarks

The `SABulkCopyColumnMapping` class is not available in the .NET Compact Framework 2.0.

## DestinationColumn property

Gets or sets the name of the column in the destination database table being mapped to.

### Visual Basic syntax

```
Public Property DestinationColumn As String
```

### C# syntax

```
public string DestinationColumn {get;set;}
```

## Remarks

A string specifying the name of the column in the destination table or a null reference (Nothing in Visual Basic) if the `DestinationOrdinal` property has priority.

The `DestinationColumn` property and `DestinationOrdinal` property are mutually exclusive. The most recently set value takes priority.

Setting the `DestinationColumn` property causes the `DestinationOrdinal` property to be set to -1. Setting the `DestinationOrdinal` property causes the `DestinationColumn` property to be set to a null reference (Nothing in Visual Basic).

It is an error to set `DestinationColumn` to null or the empty string.

**See also**

- [“SABulkCopyColumnMapping.DestinationOrdinal property \[SQL Anywhere .NET\]” on page 113](#)

## DestinationOrdinal property

Gets or sets the ordinal value of the column in the destination table being mapped to.

**Visual Basic syntax**

```
Public Property DestinationOrdinal As Integer
```

**C# syntax**

```
public int DestinationOrdinal {get;set;}
```

**Remarks**

An integer specifying the ordinal of the column being mapped to in the destination table or -1 if the property is not set.

The `DestinationColumn` property and `DestinationOrdinal` property are mutually exclusive. The most recently set value takes priority.

Setting the `DestinationColumn` property causes the `DestinationOrdinal` property to be set to -1. Setting the `DestinationOrdinal` property causes the `DestinationColumn` property to be set to a null reference (Nothing in Visual Basic).

**See also**

- [“SABulkCopyColumnMapping.DestinationColumn property \[SQL Anywhere .NET\]” on page 112](#)

## SourceColumn property

Gets or sets the name of the column being mapped in the data source.

**Visual Basic syntax**

```
Public Property SourceColumn As String
```

**C# syntax**

```
public string SourceColumn {get;set;}
```

### Remarks

A string specifying the name of the column in the data source or a null reference (Nothing in Visual Basic) if the SourceOrdinal property has priority.

The SourceColumn property and SourceOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

It is an error to set SourceColumn to null or the empty string.

### See also

- [“SABulkCopyColumnMapping.SourceOrdinal property \[SQL Anywhere .NET\]”](#) on page 114

## SourceOrdinal property

Gets or sets ordinal position of the source column within the data source.

### Visual Basic syntax

```
Public Property SourceOrdinal As Integer
```

### C# syntax

```
public int sourceOrdinal {get;set;}
```

### Remarks

An integer specifying the ordinal of the column in the data source or -1 if the property is not set.

The SourceColumn property and SourceOrdinal property are mutually exclusive. The most recently set value takes priority.

Setting the SourceColumn property causes the SourceOrdinal property to be set to -1. Setting the SourceOrdinal property causes the SourceColumn property to be set to a null reference (Nothing in Visual Basic).

### See also

- [“SABulkCopyColumnMapping.SourceColumn property \[SQL Anywhere .NET\]”](#) on page 113

## SABulkCopyColumnMappingCollection class

A collection of SABulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase.

**Visual Basic syntax**

```
Public NotInheritable Class SABulkCopyColumnMappingCollection
    Inherits System.Collections.CollectionBase
```

**C# syntax**

```
public sealed class SABulkCopyColumnMappingCollection :
    System.Collections.CollectionBase
```

**Base classes**

- [System.Collections.CollectionBase](#)

**Members**

All members of SABulkCopyColumnMappingCollection class, including all inherited members.

Name	Description
<a href="#">Add method</a>	Adds the specified SABulkCopyColumnMapping object to the collection.
<a href="#">Clear method</a> (Inherited from System.Collections.CollectionBase)	Removes all objects from the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">Contains method</a>	Gets a value indicating whether a specified SABulkCopyColumnMapping object exists in the collection.
<a href="#">CopyTo method</a>	Copies the elements of the SABulkCopyColumnMappingCollection to an array of SABulkCopyColumnMapping items, starting at a particular index.
<a href="#">GetEnumerator method</a> (Inherited from System.Collections.CollectionBase)	Returns an enumerator that iterates through the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">IndexOf method</a>	Gets or sets the index of the specified SABulkCopyColumnMapping object within the collection.
<a href="#">OnClear method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when clearing the contents of the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnClearComplete method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after clearing the contents of the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnInsert method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes before inserting a new element into the <a href="#">System.Collections.CollectionBase</a> instance.

Name	Description
<a href="#">OnInsertComplete method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after inserting a new element into the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnRemove method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when removing an element from the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnRemoveComplete method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after removing an element from the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnSet method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes before setting a value in the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnSetComplete method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes after setting a value in the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">OnValidate method</a> (Inherited from System.Collections.CollectionBase)	Performs additional custom processes when validating a value.
<a href="#">Remove method</a>	Removes the specified SABulkCopyColumnMapping element from the SABulkCopyColumnMappingCollection.
<a href="#">RemoveAt method</a>	Removes the mapping at the specified index from the collection.
<a href="#">Capacity property</a> (Inherited from System.Collections.CollectionBase)	Gets or sets the number of elements that the <a href="#">System.Collections.CollectionBase</a> can contain.
<a href="#">Count property</a> (Inherited from System.Collections.CollectionBase)	Gets the number of elements contained in the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">InnerList property</a> (Inherited from System.Collections.CollectionBase)	Gets an <a href="#">System.Collections.ArrayList</a> containing the list of elements in the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">List property</a> (Inherited from System.Collections.CollectionBase)	Gets an <a href="#">System.Collections.IList</a> containing the list of elements in the <a href="#">System.Collections.CollectionBase</a> instance.
<a href="#">this property</a>	Gets the SABulkCopyColumnMapping object at the specified index.

**Remarks**

The `SABulkCopyColumnMappingCollection` class is not available in the .NET Compact Framework 2.0.

**Add method**

Adds the specified `SABulkCopyColumnMapping` object to the collection.

**Overload list**

Name	Description
<a href="#">Add(int, int) method</a>	Creates a new <code>SABulkCopyColumnMapping</code> object using ordinals to specify both source and destination columns, and adds the mapping to the collection.
<a href="#">Add(int, string) method</a>	Creates a new <code>SABulkCopyColumnMapping</code> object using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.
<a href="#">Add(SABulkCopyColumnMapping) method</a>	Adds the specified <code>SABulkCopyColumnMapping</code> object to the collection.
<a href="#">Add(string, int) method</a>	Creates a new <code>SABulkCopyColumnMapping</code> object using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.
<a href="#">Add(string, string) method</a>	Creates a new <code>SABulkCopyColumnMapping</code> object using column names to specify both source and destination columns, and adds the mapping to the collection.

**Add(int, int) method**

Creates a new `SABulkCopyColumnMapping` object using ordinals to specify both source and destination columns, and adds the mapping to the collection.

**Visual Basic syntax**

```
Public Function Add(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumnOrdinal As Integer  
) As SABulkCopyColumnMapping
```

**C# syntax**

```
public SABulkCopyColumnMapping Add(  
    int sourceColumnOrdinal,  
    int destinationColumnOrdinal  
)
```

### Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table.

### Remarks

The `SABulkCopyColumnMappingCollection` class is not available in the .NET Compact Framework 2.0.

### Add(int, string) method

Creates a new `SABulkCopyColumnMapping` object using a column ordinal to refer to the source column and a column name to refer to the destination column, and adds mapping to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal sourceColumnOrdinal As Integer,  
    ByVal destinationColumn As String  
) As SABulkCopyColumnMapping
```

### C# syntax

```
public SABulkCopyColumnMapping Add(  
    int sourceColumnOrdinal,  
    string destinationColumn  
)
```

### Parameters

- **sourceColumnOrdinal** The ordinal position of the source column within the data source.
- **destinationColumn** The name of the destination column within the destination table.

### Remarks

The `SABulkCopyColumnMappingCollection` class is not available in the .NET Compact Framework 2.0.

### Add(SABulkCopyColumnMapping) method

Adds the specified `SABulkCopyColumnMapping` object to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal bulkCopyColumnMapping As SABulkCopyColumnMapping  
) As SABulkCopyColumnMapping
```

### C# syntax

```
public SABulkCopyColumnMapping Add(  
    SABulkCopyColumnMapping bulkCopyColumnMapping  
)
```



**Parameters**

- **bulkCopyColumnMapping** The SABulkCopyColumnMapping object that describes the mapping to be added to the collection.

**Remarks**

The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

**See also**

- [“SABulkCopyColumnMapping class \[SQL Anywhere .NET\]” on page 108](#)

**Add(string, int) method**

Creates a new SABulkCopyColumnMapping object using a column name to refer to the source column and a column ordinal to refer to the destination the column, and adds the mapping to the collection.

**Visual Basic syntax**

```
Public Function Add(  
    ByVal sourceColumn As String,  
    ByVal destinationColumnOrdinal As Integer  
) As SABulkCopyColumnMapping
```

**C# syntax**

```
public SABulkCopyColumnMapping Add(  
    string sourceColumn,  
    int destinationColumnOrdinal  
)
```

**Parameters**

- **sourceColumn** The name of the source column within the data source.
- **destinationColumnOrdinal** The ordinal position of the destination column within the destination table.

**Remarks**

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

**Add(string, string) method**

Creates a new SABulkCopyColumnMapping object using column names to specify both source and destination columns, and adds the mapping to the collection.

**Visual Basic syntax**

```
Public Function Add(  
    ByVal sourceColumn As String,
```

```
        ByVal destinationColumn As String  
    ) As SABulkCopyColumnMapping
```

### C# syntax

```
public SABulkCopyColumnMapping Add(  
    string sourceColumn,  
    string destinationColumn  
)
```

### Parameters

- **sourceColumn** The name of the source column within the data source.
- **destinationColumn** The name of the destination column within the destination table.

### Remarks

The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

## Contains method

Gets a value indicating whether a specified SABulkCopyColumnMapping object exists in the collection.

### Visual Basic syntax

```
Public Function Contains(  
    ByVal value As SABulkCopyColumnMapping  
) As Boolean
```

### C# syntax

```
public bool Contains(SABulkCopyColumnMapping value)
```

### Parameters

- **value** A valid SABulkCopyColumnMapping object.

### Returns

True if the specified mapping exists in the collection; otherwise, false.

## CopyTo method

Copies the elements of the SABulkCopyColumnMappingCollection to an array of SABulkCopyColumnMapping items, starting at a particular index.

### Visual Basic syntax

```
Public Sub CopyTo(  
    ByVal array As SABulkCopyColumnMapping(),  
    ByVal index As Integer  
)
```

**C# syntax**

```
public void CopyTo(SABulkCopyColumnMapping[] array, int index)
```

**Parameters**

- **array** The one-dimensional SABulkCopyColumnMapping array that is the destination of the elements copied from SABulkCopyColumnMappingCollection. The array must have zero-based indexing.
- **index** The zero-based index in the array at which copying begins.

## IndexOf method

Gets or sets the index of the specified SABulkCopyColumnMapping object within the collection.

**Visual Basic syntax**

```
Public Function IndexOf(  
    ByVal value As SABulkCopyColumnMapping  
    ) As Integer
```

**C# syntax**

```
public int IndexOf(SABulkCopyColumnMapping value)
```

**Parameters**

- **value** The SABulkCopyColumnMapping object to search for.

**Returns**

The zero-based index of the column mapping is returned, or -1 is returned if the column mapping is not found in the collection.

## Remove method

Removes the specified SABulkCopyColumnMapping element from the SABulkCopyColumnMappingCollection.

**Visual Basic syntax**

```
Public Sub Remove(ByVal value As SABulkCopyColumnMapping)
```

**C# syntax**

```
public void Remove(SABulkCopyColumnMapping value)
```

**Parameters**

- **value** The SABulkCopyColumnMapping object to be removed from the collection.

## RemoveAt method

Removes the mapping at the specified index from the collection.

### Visual Basic syntax

```
Public Shadows Sub RemoveAt(ByVal index As Integer)
```

### C# syntax

```
public new void RemoveAt(int index)
```

### Parameters

- **index** The zero-based index of the SABulkCopyColumnMapping object to be removed from the collection.

## this property

Gets the SABulkCopyColumnMapping object at the specified index.

### Visual Basic syntax

```
Public ReadOnly Property Item(  
    ByVal index As Integer  
) As SABulkCopyColumnMapping
```

### C# syntax

```
public SABulkCopyColumnMapping this[int index] {get;}
```

### Parameters

- **index** The zero-based index of the SABulkCopyColumnMapping object to find.

### Returns

An SABulkCopyColumnMapping object is returned.

## SACommand class

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

### Visual Basic syntax

```
Public NotInheritable Class SACommand  
    Inherits System.Data.Common.DbCommand  
    Implements System.ICloneable
```

**C# syntax**

```
public sealed class SACommand :
    System.Data.Common.DbCommand,
    System.ICloneable
```

**Base classes**

- [System.Data.Common.DbCommand](#)
- [System.ICloneable](#)

**Members**

All members of SACommand class, including all inherited members.

Name	Description
<a href="#">SACommand constructor</a>	Initializes an SACommand object.
<a href="#">BeginExecuteNonQuery method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.
<a href="#">BeginExecuteReader method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand, and retrieves one or more result sets from the database server.
<a href="#">Cancel method</a>	Cancels the execution of an SACommand object.
<a href="#">CreateParameter method</a>	Provides an SAParameter object for supplying parameters to SACommand objects.
<a href="#">EndExecuteNonQuery method</a>	Finishes asynchronous execution of a SQL statement or stored procedure.
<a href="#">EndExecuteReader method</a>	Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested SADATAReader.
<a href="#">ExecuteDbDataReader method</a> (Inherited from System.Data.Common.DbCommand)	Executes the command text against the connection.
<a href="#">ExecuteNonQuery method</a>	Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.
<a href="#">ExecuteReader method</a>	Executes a SQL statement that returns a result set.
<a href="#">ExecuteScalar method</a>	Executes a statement that returns a single value.
<a href="#">Prepare method</a>	Prepares or compiles the SACommand on the data source.

Name	Description
<a href="#">ResetCommandTimeout method</a>	Resets the CommandTimeout property to its default value of 30 seconds.
<a href="#">CommandText property</a>	Gets or sets the text of a SQL statement or stored procedure.
<a href="#">CommandTimeout property</a>	This feature is not supported by the SQL Anywhere .NET Data Provider.
<a href="#">CommandType property</a>	Gets or sets the type of command represented by an SACommand.
<a href="#">Connection property</a>	Gets or sets the connection object to which the SACommand object applies.
<a href="#">DesignTimeVisible property</a>	Gets or sets a value that indicates if the SACommand should be visible in a Windows Form Designer control.
<a href="#">Parameters property</a>	A collection of parameters for the current statement.
<a href="#">Transaction property</a>	Specifies the SATransaction object in which the SACommand executes.
<a href="#">UpdatedRowSource property</a>	Gets or sets how command results are applied to the DataRow when used by the Update method of the SADataAdapter.

**Remarks**

**Implements:** ICloneable

For more information, see [“Data access and manipulation”](#) on page 46.

## SACommand constructor

Initializes an SACommand object.

**Overload list**

Name	Description
<a href="#">SACommand() constructor</a>	Initializes an SACommand object.
<a href="#">SACommand(string) constructor</a>	Initializes an SACommand object.
<a href="#">SACommand(string, SAConnection) constructor</a>	A SQL statement or stored procedure that is executed against a SQL Anywhere database.

Name	Description
<a href="#">SACommand(string, SAConnection, SATransaction) constructor</a>	A SQL statement or stored procedure that is executed against a SQL Anywhere database.

## SACommand() constructor

Initializes an SACommand object.

### Visual Basic syntax

```
Public Sub New()
```

### C# syntax

```
public SACommand()
```

## SACommand(string) constructor

Initializes an SACommand object.

### Visual Basic syntax

```
Public Sub New(ByVal cmdText As String)
```

### C# syntax

```
public SACommand(string cmdText)
```

### Parameters

- **cmdText** The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.

## SACommand(string, SAConnection) constructor

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

### Visual Basic syntax

```
Public Sub New(  
    ByVal cmdText As String,  
    ByVal connection As SAConnection  
)
```

### C# syntax

```
public SACommand(string cmdText, SAConnection connection)
```

### Parameters

- **cmdText** The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.

- **connection** The current connection.

## SACommand(string, SAConnection, SATransaction) constructor

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

### Visual Basic syntax

```
Public Sub New(  
    ByVal cmdText As String,  
    ByVal connection As SAConnection,  
    ByVal transaction As SATransaction  
)
```

### C# syntax

```
public SACommand(  
    string cmdText,  
    SAConnection connection,  
    SATransaction transaction  
)
```

### Parameters

- **cmdText** The text of the SQL statement or stored procedure. For parameterized statements, use a question mark (?) placeholder to pass parameters.
- **connection** The current connection.
- **transaction** The SATransaction object in which the SAConnection executes.

### See also

- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)

## BeginExecuteNonQuery method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.

### Overload list

Name	Description
<a href="#">BeginExecuteNonQuery() method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.
<a href="#">BeginExecuteNon-Query(AsyncCallback, object) method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand, given a callback procedure and state information.



## BeginExecuteNonQuery() method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this `SACCommand`.

### Visual Basic syntax

```
Public Function BeginExecuteNonQuery() As IAsyncResult
```

### C# syntax

```
public IAsyncResult BeginExecuteNonQuery()
```

### Returns

An `System.IAsyncResult` that can be used to poll, wait for results, or both; this value is also needed when invoking `EndExecuteNonQuery(IAsyncResult)`, which returns the number of affected rows.

### Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

### Remarks

For asynchronous command, the order of parameters must be consistent with `CommandText`.

### See also

- [“SACCommand.EndExecuteNonQuery method \[SQL Anywhere .NET\]” on page 133](#)

## BeginExecuteNonQuery(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this `SACCommand`, given a callback procedure and state information.

### Visual Basic syntax

```
Public Function BeginExecuteNonQuery(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object  
) As IAsyncResult
```

### C# syntax

```
public IAsyncResult BeginExecuteNonQuery(  
    AsyncCallback callback,  
    object stateObject  
)
```

### Parameters

- **callback** An `System.AsyncCallback` delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.
- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the `System.IAsyncResult.AsyncState` property.

### Returns

An System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteNonQuery(IAsyncResult), which returns the number of affected rows.

### Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

### Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

### See also

- [“SACCommand.EndExecuteNonQuery method \[SQL Anywhere .NET\]” on page 133](#)

## BeginExecuteReader method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACCommand, and retrieves one or more result sets from the database server.

### Overload list

Name	Description
<a href="#">BeginExecuteReader() method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACCommand, and retrieves one or more result sets from the database server.
<a href="#">BeginExecuteReader(AsyncCallback, object) method</a>	Initiates the asynchronous execution of a SQL statement that is described by the SACCommand object, and retrieves the result set, given a callback procedure and state information.
<a href="#">BeginExecuteReader(AsyncCallback, object, CommandBehavior) method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACCommand, and retrieves one or more result sets from the server.
<a href="#">BeginExecuteReader(CommandBehavior) method</a>	Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACCommand, and retrieves one or more result sets from the server.

### BeginExecuteReader() method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACCommand, and retrieves one or more result sets from the database server.

### Visual Basic syntax

```
Public Function BeginExecuteReader() As IAsyncResult
```

### C# syntax

```
public IAsyncResult BeginExecuteReader()
```

### Returns

An System.IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an SADataReader object that can be used to retrieve the returned rows.

### Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

### Remarks

For asynchronous command, the order of parameters must be consistent with CommandText.

### See also

- [“SACommand.EndExecuteReader method \[SQL Anywhere .NET\]” on page 135](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)

## BeginExecuteReader(AsyncCallback, object) method

Initiates the asynchronous execution of a SQL statement that is described by the SACommand object, and retrieves the result set, given a callback procedure and state information.

### Visual Basic syntax

```
Public Function BeginExecuteReader(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object  
) As IAsyncResult
```

### C# syntax

```
public IAsyncResult BeginExecuteReader(  
    AsyncCallback callback,  
    object stateObject  
)
```

### Parameters

- **callback** An System.AsyncCallback delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.
- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the System.IAsyncResult.AsyncState property.

## Returns

An `System.IAsyncResult` that can be used to poll, wait for results, or both; this value is also needed when invoking `EndExecuteReader(IAsyncResult)`, which returns an `SADataReader` object that can be used to retrieve the returned rows.

## Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

## Remarks

For asynchronous command, the order of parameters must be consistent with `CommandText`.

## See also

- [“SACommand.EndExecuteReader method \[SQL Anywhere .NET\]” on page 135](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)

## BeginExecuteReader(AsyncCallback, object, CommandBehavior) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this `SACommand`, and retrieves one or more result sets from the server.

## Visual Basic syntax

```
Public Function BeginExecuteReader(  
    ByVal callback As AsyncCallback,  
    ByVal stateObject As Object,  
    ByVal behavior As CommandBehavior  
) As IAsyncResult
```

## C# syntax

```
public IAsyncResult BeginExecuteReader(  
    AsyncCallback callback,  
    object stateObject,  
    CommandBehavior behavior  
)
```

## Parameters

- **callback** An `System.AsyncCallback` delegate that is invoked when the command's execution has completed. Pass null (Nothing in Microsoft Visual Basic) to indicate that no callback is required.
- **stateObject** A user-defined state object that is passed to the callback procedure. Retrieve this object from within the callback procedure using the `System.IAsyncResult.AsyncState` property.
- **behavior** A bitwise combination of `System.Data.CommandBehavior` flags describing the results of the query and its effect on the connection.

## Returns

An `System.IAsyncResult` that can be used to poll, wait for results, or both; this value is also needed when invoking `EndExecuteReader(IAsyncResult)`, which returns an `SADataReader` object that can be used to retrieve the returned rows.

## Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

## Remarks

For asynchronous command, the order of parameters must be consistent with `CommandText`.

## See also

- [“SACommand.EndExecuteReader method \[SQL Anywhere .NET\]” on page 135](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)

## BeginExecuteReader(CommandBehavior) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this `SACommand`, and retrieves one or more result sets from the server.

## Visual Basic syntax

```
Public Function BeginExecuteReader(  
    ByVal behavior As CommandBehavior  
) As IAsyncResult
```

## C# syntax

```
public IAsyncResult BeginExecuteReader(CommandBehavior behavior)
```

## Parameters

- **behavior** A bitwise combination of `System.Data.CommandBehavior` flags describing the results of the query and its effect on the connection.

## Returns

An `System.IAsyncResult` that can be used to poll, wait for results, or both; this value is also needed when invoking `EndExecuteReader(IAsyncResult)`, which returns an `SADataReader` object that can be used to retrieve the returned rows.

## Exceptions

- [SAException class](#) Any error that occurred while executing the command text.

## Remarks

For asynchronous command, the order of parameters must be consistent with `CommandText`.

### See also

- [“SACCommand.EndExecuteReader method \[SQL Anywhere .NET\]” on page 135](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)

## Cancel method

Cancels the execution of an SACCommand object.

### Visual Basic syntax

```
Public Overrides Sub Cancel()
```

### C# syntax

```
public override void Cancel()
```

### Remarks

If there is nothing to cancel, nothing happens. If there is a command in process, a "Statement interrupted by user" exception is thrown.

## CreateParameter method

Provides an SAParameter object for supplying parameters to SACCommand objects.

### Visual Basic syntax

```
Public Shadows Function CreateParameter() As SAParameter
```

### C# syntax

```
public new SAParameter CreateParameter()
```

### Returns

A new parameter, as an SAParameter object.

### Remarks

Stored procedures and some other SQL statements can take parameters, indicated in the text of a statement by a question mark (?).

The CreateParameter method provides an SAParameter object. You can set properties on the SAParameter to specify the value, data type, and so on for the parameter.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)

## EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement or stored procedure.

### Visual Basic syntax

```
Public Function EndExecuteNonQuery(
    ByVal asyncResult As IAsyncResult
) As Integer
```

### C# syntax

```
public int EndExecuteNonQuery(IAsyncResult asyncResult)
```

### Parameters

- **asyncResult** The IAsyncResult returned by the call to `SACCommand.BeginExecuteNonQuery`.

### Returns

The number of rows affected (the same behavior as `SACCommand.ExecuteNonQuery`).

### Exceptions

- **ArgumentException** The `asyncResult` parameter is null (Nothing in Microsoft Visual Basic).
- **InvalidOperationException** The `SACCommand.EndExecuteNonQuery(IAsyncResult)` was called more than once for a single command execution, or the method was mismatched against its execution method.

### Remarks

You must call `EndExecuteNonQuery` once for every call to `BeginExecuteNonQuery`. The call must be after `BeginExecuteNonQuery` has returned. ADO.NET is not thread safe; it is your responsibility to ensure that `BeginExecuteNonQuery` has returned. The `IAsyncResult` passed to `EndExecuteNonQuery` must be the same as the one returned from the `BeginExecuteNonQuery` call that is being completed. It is an error to call `EndExecuteNonQuery` to end a call to `BeginExecuteReader`, and vice versa.

If an error occurs while executing the command, the exception is thrown when `EndExecuteNonQuery` is called.

There are four ways to wait for execution to complete:

- (1) Call `EndExecuteNonQuery`.

Calling `EndExecuteNonQuery` blocks until the command completes. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
```

```
// this will block until the command completes
int rowCount reader = cmd.EndExecuteNonQuery( res );
```

(2) Poll the IsCompleted property of the IAsyncResult.

You can poll the IsCompleted property of the IAsyncResult. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
while( !res.IsCompleted ) {
    // do other work
}
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(3) Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object.

You can use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'"
    + " WHERE DepartmentID=100",
    conn );
IAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this will not block because the command is finished
int rowCount = cmd.EndExecuteNonQuery( res );
```

(4) Specify a callback function when calling BeginExecuteNonQuery.

You can specify a callback function when calling BeginExecuteNonQuery. For example:

```
private void callbackFunction( IAsyncResult ar ) {
    SACCommand cmd = (SACCommand) ar.AsyncState;
    // this won't block since the command has completed
    int rowCount = cmd.EndExecuteNonQuery();
}

// elsewhere in the code
private void DoStuff() {
    SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
    conn.Open();
    SACCommand cmd = new SACCommand(
        "UPDATE Departments"
        + " SET DepartmentName = 'Engineering'"
        + " WHERE DepartmentID=100",
        conn );
    IAsyncResult res = cmd.BeginExecuteNonQuery( callbackFunction, cmd );
    // perform other work. The callback function will be
```



```
    } // called when the command completes
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

### See also

- [“SACCommand.BeginExecuteNonQuery method \[SQL Anywhere .NET\]” on page 126](#)

## EndExecuteReader method

Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested `SADataReader`.

### Visual Basic syntax

```
Public Function EndExecuteReader(  
    ByVal asyncResult As IAsyncResult  
) As SADataReader
```

### C# syntax

```
public SADataReader EndExecuteReader(IAsyncResult asyncResult)
```

### Parameters

- **asyncResult** The `IAsyncResult` returned by the call to `SACCommand.BeginExecuteReader`.

### Returns

An `SADataReader` object that can be used to retrieve the requested rows (the same behavior as `SACCommand.ExecuteReader`).

### Exceptions

- **ArgumentException** The `asyncResult` parameter is null (Nothing in Microsoft Visual Basic)
- **InvalidOperationException** The `SACCommand.EndExecuteReader(IAsyncResult)` was called more than once for a single command execution, or the method was mismatched against its execution method.

### Remarks

You must call `EndExecuteReader` once for every call to `BeginExecuteReader`. The call must be after `BeginExecuteReader` has returned. ADO.NET is not thread safe; it is your responsibility to ensure that `BeginExecuteReader` has returned. The `IAsyncResult` passed to `EndExecuteReader` must be the same as the one returned from the `BeginExecuteReader` call that is being completed. It is an error to call `EndExecuteReader` to end a call to `BeginExecuteNonQuery`, and vice versa.

If an error occurs while executing the command, the exception is thrown when `EndExecuteReader` is called.

There are four ways to wait for execution to complete:

(1) Call EndExecuteReader.

Calling EndExecuteReader blocks until the command completes. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this blocks until the command completes
SADataReader reader = cmd.EndExecuteReader( res );
```

(2) Poll the IsCompleted property of the IAsyncResult.

You can poll the IsCompleted property of the IAsyncResult. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
while( !res.IsCompleted ) {
    // do other work
}
// this does not block because the command is finished
SADataReader reader = cmd.EndExecuteReader( res );
```

(3) Use the IAsyncResult.AsyncWaitHandle property to get a synchronization object.

You can use the IAsyncResult.AsyncWaitHandle property to get a synchronization object, and wait on that. For example:

```
SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
conn.Open();
SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this does not block because the command is finished
SADataReader reader = cmd.EndExecuteReader( res );
```

(4) Specify a callback function when calling BeginExecuteReader

You can specify a callback function when calling BeginExecuteReader. For example:

```
private void callbackFunction( IAsyncResult ar ) {
    SACCommand cmd = (SACCommand) ar.AsyncState;
    // this does not block since the command has completed
    SADataReader reader = cmd.EndExecuteReader();
}

// elsewhere in the code
private void DoStuff() {
    SACConnection conn = new SACConnection("DSN=SQL Anywhere 12 Demo");
    conn.Open();
    SACCommand cmd = new SACCommand( "SELECT * FROM Departments", conn );
    IAsyncResult res = cmd.BeginExecuteReader( callbackFunction, cmd );
    // perform other work. The callback function will be
    // called when the command completes
}
```

The callback function executes in a separate thread, so the usual caveats related to updating the user interface in a threaded program apply.

#### See also

- [“SACommand.BeginExecuteReader method \[SQL Anywhere .NET\]” on page 128](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)

## ExecuteNonQuery method

Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.

#### Visual Basic syntax

```
Public Overrides Function ExecuteNonQuery() As Integer
```

#### C# syntax

```
public override int ExecuteNonQuery()
```

#### Returns

The number of rows affected.

#### Remarks

You can use ExecuteNonQuery to change the data in a database without using a DataSet. Do this by executing UPDATE, INSERT, or DELETE statements.

Although ExecuteNonQuery does not return any rows, output parameters or return values that are mapped to parameters are populated with data.

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, and for rollbacks, the return value is -1.

#### See also

- [“SACommand.ExecuteReader method \[SQL Anywhere .NET\]” on page 137](#)

## ExecuteReader method

Executes a SQL statement that returns a result set.

#### Overload list

Name	Description
<a href="#">ExecuteReader() method</a>	Executes a SQL statement that returns a result set.

Name	Description
<a href="#">ExecuteReader(CommandBehavior) method</a>	Executes a SQL statement that returns a result set.

## ExecuteReader() method

Executes a SQL statement that returns a result set.

### Visual Basic syntax

```
Public Shadows Function ExecuteReader() As SqlDataReader
```

### C# syntax

```
public new SqlDataReader ExecuteReader()
```

### Returns

The result set as an SqlDataReader object.

### Remarks

The statement is the current SACommand object, with CommandText and Parameters as needed. The SqlDataReader object is a read-only, forward-only result set. For modifiable result sets, use an SqlDataAdapter.

### See also

- [“SACommand.ExecuteNonQuery method \[SQL Anywhere .NET\]” on page 137](#)
- [“SqlDataReader class \[SQL Anywhere .NET\]” on page 214](#)
- [“SqlDataAdapter class \[SQL Anywhere .NET\]” on page 205](#)
- [“SACommand.CommandText property \[SQL Anywhere .NET\]” on page 141](#)
- [“SACommand.Parameters property \[SQL Anywhere .NET\]” on page 143](#)

## ExecuteReader(CommandBehavior) method

Executes a SQL statement that returns a result set.

### Visual Basic syntax

```
Public Shadows Function ExecuteReader(  
    ByVal behavior As CommandBehavior  
) As SqlDataReader
```

### C# syntax

```
public new SqlDataReader ExecuteReader(CommandBehavior behavior)
```

### Parameters

- **behavior** One of CloseConnection, Default, KeyInfo, SchemaOnly, SequentialAccess, SingleResult, or SingleRow. For more information about this parameter, see the .NET Framework documentation for CommandBehavior Enumeration.

**Returns**

The result set as an `SADataReader` object.

**Remarks**

The statement is the current `SACCommand` object, with `CommandText` and `Parameters` as needed. The `SADataReader` object is a read-only, forward-only result set. For modifiable result sets, use an `SADataAdapter`.

**See also**

- [“SACCommand.ExecuteNonQuery method \[SQL Anywhere .NET\]” on page 137](#)
- [“SADataReader class \[SQL Anywhere .NET\]” on page 214](#)
- [“SADataAdapter class \[SQL Anywhere .NET\]” on page 205](#)
- [“SACCommand.CommandText property \[SQL Anywhere .NET\]” on page 141](#)
- [“SACCommand.Parameters property \[SQL Anywhere .NET\]” on page 143](#)

## ExecuteScalar method

Executes a statement that returns a single value.

**Visual Basic syntax**

```
Public Overrides Function ExecuteScalar() As Object
```

**C# syntax**

```
public override object ExecuteScalar()
```

**Returns**

The first column of the first row in the result set, or a null reference if the result set is empty.

**Remarks**

If this method is called on a query that returns multiple rows and columns, only the first column of the first row is returned.

## Prepare method

Prepares or compiles the `SACCommand` on the data source.

**Visual Basic syntax**

```
Public Overrides Sub Prepare()
```

**C# syntax**

```
public override void Prepare()
```

## Remarks

If you call one of the `ExecuteNonQuery`, `ExecuteReader`, or `ExecuteScalar` methods after calling `Prepare`, any parameter value that is larger than the value specified by the `Size` property is automatically truncated to the original specified size of the parameter, and no truncation errors are returned.

The truncation only happens for the following data types:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- BINARY
- LONG BINARY
- VARBINARY
- IMAGE

If the `size` property is not specified, and so is using the default value, the data is not truncated.

## See also

- [“SACCommand.ExecuteNonQuery method \[SQL Anywhere .NET\]” on page 137](#)
- [“SACCommand.ExecuteReader method \[SQL Anywhere .NET\]” on page 137](#)
- [“SACCommand.ExecuteScalar method \[SQL Anywhere .NET\]” on page 139](#)

## ResetCommandTimeout method

Resets the `CommandTimeout` property to its default value of 30 seconds.

### Visual Basic syntax

```
Public Sub ResetCommandTimeout()
```

### C# syntax

```
public void ResetCommandTimeout()
```

## CommandText property

Gets or sets the text of a SQL statement or stored procedure.

### Visual Basic syntax

```
Public Overrides Property CommandText As String
```

### C# syntax

```
public override string CommandText {get;set;}
```

### Remarks

The SQL statement or the name of the stored procedure to execute. The default is an empty string.

### See also

- [“SACommand.SACommand constructor \[SQL Anywhere .NET\]” on page 124](#)

## CommandTimeout property

This feature is not supported by the SQL Anywhere .NET Data Provider.

### Visual Basic syntax

```
Public Overrides Property CommandTimeout As Integer
```

### C# syntax

```
public override int CommandTimeout {get;set;}
```

### Remarks

See the following example to set a request timeout:

```
cmd.CommandText = "SET OPTION request_timeout = 30";  
cmd.ExecuteNonQuery();
```

## CommandType property

Gets or sets the type of command represented by an SACommand.

### Visual Basic syntax

```
Public Overrides Property CommandType As CommandType
```

### C# syntax

```
public override CommandType CommandType {get;set;}
```

### Remarks

One of the System.Data.CommandType values. The default is System.Data.CommandType.Text.

Supported command types are as follows:

- `System.Data.CommandType.StoredProcedure` When you specify this `CommandType`, the command text must be the name of a stored procedure and you must supply any arguments as `SAParameter` objects.
- `System.Data.CommandType.Text` This is the default value.

When the `CommandType` property is set to `StoredProcedure`, the `CommandText` property should be set to the name of the stored procedure. The command executes this stored procedure when you call one of the `Execute` methods.

Use a question mark (?) placeholder to pass parameters. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which `SAParameter` objects are added to the `SAParameterCollection` must directly correspond to the position of the question mark placeholder for the parameter.

## Connection property

Gets or sets the connection object to which the `SACommand` object applies.

### Visual Basic syntax

```
Public Shadows Property Connection As SAConnection
```

### C# syntax

```
public new SAConnection Connection {get;set;}
```

### Remarks

The default value is a null reference. In Visual Basic it is `Nothing`.

## DesignTimeVisible property

Gets or sets a value that indicates if the `SACommand` should be visible in a Windows Form Designer control.

### Visual Basic syntax

```
Public Overrides Property DesignTimeVisible As Boolean
```

### C# syntax

```
public override bool DesignTimeVisible {get;set;}
```

### Remarks

The default is `true`.



True if this SACommand instance should be visible, false if this instance should not be visible. The default is false.

## Parameters property

A collection of parameters for the current statement.

### Visual Basic syntax

```
Public ReadOnly Shadows Property Parameters As SAPParameterCollection
```

### C# syntax

```
public new SAPParameterCollection Parameters {get;}
```

### Remarks

Use question marks in the CommandText to indicate parameters.

The parameters of the SQL statement or stored procedure. The default value is an empty collection.

When CommandType is set to Text, pass parameters using the question mark placeholder. For example:

```
SELECT * FROM Customers WHERE ID = ?
```

The order in which SAPParameter objects are added to the SAPParameterCollection must directly correspond to the position of the question mark placeholder for the parameter in the command text.

When the parameters in the collection do not match the requirements of the query to be executed, an error may result or an exception may be thrown.

### See also

- [“SAPParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)

## Transaction property

Specifies the SATransaction object in which the SACommand executes.

### Visual Basic syntax

```
Public Shadows Property Transaction As SATransaction
```

### C# syntax

```
public new SATransaction Transaction {get;set;}
```

### Remarks

The default value is a null reference. In Visual Basic, this is Nothing.

You cannot set the Transaction property if it is already set to a specific value and the command is executing. If you set the transaction property to an SATransaction object that is not connected to the same

SACConnection object as the SACCommand object, an exception will be thrown the next time you attempt to execute a statement.

For more information, see [“Transaction processing” on page 62](#).

#### See also

- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)

## UpdatedRowSource property

Gets or sets how command results are applied to the DataRow when used by the Update method of the SDataAdapter.

#### Visual Basic syntax

```
Public Overrides Property UpdatedRowSource As UpdateRowSource
```

#### C# syntax

```
public override UpdateRowSource UpdatedRowSource {get;set;}
```

#### Remarks

One of the UpdatedRowSource values. The default value is UpdateRowSource.OutputParameters. If the command is automatically generated, this property is UpdateRowSource.None.

UpdatedRowSource.Both, which returns both resultset and output parameters, is not supported.

## SACCommandBuilder class

A way to generate single-table SQL statements that reconcile changes made to a DataSet with the data in the associated database.

#### Visual Basic syntax

```
Public NotInheritable Class SACCommandBuilder  
    Inherits System.Data.Common.DbCommandBuilder
```

#### C# syntax

```
public sealed class SACCommandBuilder :  
    System.Data.Common.DbCommandBuilder
```

#### Base classes

- [System.Data.Common.DbCommandBuilder](#)

#### Members

All members of SACCommandBuilder class, including all inherited members.

Name	Description
<a href="#">SACommandBuilder</a> constructor	Initializes an SACommandBuilder object.
<a href="#">ApplyParameterInfo</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Allows the provider implementation of the <a href="#">System.Data.Common.DbCommandBuilder</a> class to handle additional parameter properties.
<a href="#">DeriveParameters</a> method	Populates the Parameters collection of the specified SACommand object.
<a href="#">Dispose</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Releases the unmanaged resources used by the <a href="#">System.Data.Common.DbCommandBuilder</a> and optionally releases the managed resources.
<a href="#">GetDeleteCommand</a> method	Returns the generated SACommand object that performs DELETE operations on the database when SADataAdapter.Update is called.
<a href="#">GetInsertCommand</a> method	Returns the generated SACommand object that performs INSERT operations on the database when an Update is called.
<a href="#">GetParameterName</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the name of the specified parameter in the format of @p#.
<a href="#">GetParameterPlaceholder</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the placeholder for the parameter in the associated SQL statement.
<a href="#">GetSchemaTable</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Returns the schema table for the <a href="#">System.Data.Common.DbCommandBuilder</a> .
<a href="#">GetUpdateCommand</a> method	Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.
<a href="#">InitializeCommand</a> method (Inherited from System.Data.Common.DbCommandBuilder)	Resets the <a href="#">System.Data.Common.DbCommand.CommandTimeout</a> , <a href="#">System.Data.Common.DbCommand.Transaction</a> , <a href="#">System.Data.Common.DbCommand.CommandType</a> , and <a href="#">System.Data.UpdateRowSource</a> properties on the <a href="#">System.Data.Common.DbCommand</a> .
<a href="#">QuoteIdentifier</a> method	Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

Name	Description
<a href="#">RefreshSchema method</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Clears the commands associated with this <code>System.Data.Common.DbCommandBuilder</code> .
<a href="#">RowUpdatingHandler method</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Adds an event handler for the <code>System.Data.OleDb.OleDbDataAdapter.RowUpdating</code> event.
<a href="#">SetRowUpdatingHandler method</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Registers the <code>System.Data.Common.DbCommandBuilder</code> to handle the <code>System.Data.OleDb.OleDbDataAdapter.RowUpdating</code> event for a <code>System.Data.Common.DbDataAdapter</code>
<a href="#">UnquoteIdentifier method</a>	Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.
<a href="#">CatalogLocation property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Sets or gets the <code>System.Data.Common.CatalogLocation</code> for an instance of the <code>System.Data.Common.DbCommandBuilder</code> class.
<a href="#">CatalogSeparator property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Sets or gets a string used as the catalog separator for an instance of the <code>System.Data.Common.DbCommandBuilder</code> class.
<a href="#">ConflictOption property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Specifies which <code>System.Data.ConflictOption</code> is to be used by the <code>System.Data.Common.DbCommandBuilder</code> .
<a href="#">DataAdapter property</a>	Specifies the <code>SADDataAdapter</code> for which to generate statements.
<a href="#">QuotePrefix property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Gets or sets the beginning character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.
<a href="#">QuoteSuffix property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Gets or sets the beginning character or characters to use when specifying database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.
<a href="#">SchemaSeparator property</a> (Inherited from <code>System.Data.Common.DbCommandBuilder</code> )	Gets or sets the character to be used for the separator between the schema identifier and any other identifiers.

Name	Description
<a href="#">SetAllValues property</a> (Inherited from System.Data.Common.DbCommandBuilder)	Specifies whether all column values in an update statement are included or only changed ones.

## SACommandBuilder constructor

Initializes an SACommandBuilder object.

### Overload list

Name	Description
<a href="#">SACommandBuilder() constructor</a>	Initializes an SACommandBuilder object.
<a href="#">SACommandBuilder(SADataAdapter) constructor</a>	Initializes an SACommandBuilder object.

### SACommandBuilder() constructor

Initializes an SACommandBuilder object.

#### Visual Basic syntax

```
Public Sub New()
```

#### C# syntax

```
public SACommandBuilder()
```

### SACommandBuilder(SADataAdapter) constructor

Initializes an SACommandBuilder object.

#### Visual Basic syntax

```
Public Sub New(ByVal adapter As SADataAdapter)
```

#### C# syntax

```
public SACommandBuilder(SADataAdapter adapter)
```

#### Parameters

- **adapter** An SADataAdapter object for which to generate reconciliation statements.

## DeriveParameters method

Populates the Parameters collection of the specified SACCommand object.

### Visual Basic syntax

```
Public Shared Sub DeriveParameters(ByVal command As SACCommand)
```

### C# syntax

```
public static void DeriveParameters(SACCommand command)
```

### Parameters

- **command** An SACCommand object for which to derive parameters.

### Remarks

This is used for the stored procedure specified in the SACCommand.

DeriveParameters overwrites any existing parameter information for the SACCommand.

DeriveParameters requires an extra call to the database server. If the parameter information is known in advance, it is more efficient to populate the Parameters collection by setting the information explicitly.

## GetDeleteCommand method

Returns the generated SACCommand object that performs DELETE operations on the database when SADATAAdapter.Update is called.

### Overload list

Name	Description
<a href="#">GetDeleteCommand() method</a>	Returns the generated SACCommand object that performs DELETE operations on the database when SADATAAdapter.Update is called.
<a href="#">GetDeleteCommand(bool) method</a>	Returns the generated SACCommand object that performs DELETE operations on the database when SADATAAdapter.Update is called.

### GetDeleteCommand() method

Returns the generated SACCommand object that performs DELETE operations on the database when SADATAAdapter.Update is called.

### Visual Basic syntax

```
Public Shadows Function GetDeleteCommand() As SACCommand
```

**C# syntax**

```
public new SACommand GetDeleteCommand()
```

**Returns**

The automatically generated SACommand object required to perform deletions.

**Remarks**

The GetDeleteCommand method returns the SACommand object to be executed, so it may be useful for informational or troubleshooting purposes.

You can also use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the CommandTimeout value, and then explicitly set that value on the SADataAdapter.

SQL statements are first generated when the application calls Update or GetDeleteCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetDeleteCommand will still be using information from the previous statement.

**See also**

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

**GetDeleteCommand(bool) method**

Returns the generated SACommand object that performs DELETE operations on the database when SADataAdapter.Update is called.

**Visual Basic syntax**

```
Public Shadows Function GetDeleteCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

**C# syntax**

```
public new SACommand GetDeleteCommand(bool useColumnsForParameterNames)
```

**Parameters**

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

**Returns**

The automatically generated SACommand object required to perform deletions.

**Remarks**

The GetDeleteCommand method returns the SACommand object to be executed, so it may be useful for informational or troubleshooting purposes.

You can also use `GetDeleteCommand` as the basis of a modified command. For example, you might call `GetDeleteCommand` and modify the `CommandTimeout` value, and then explicitly set that value on the `SADataAdapter`.

SQL statements are first generated when the application calls `Update` or `GetDeleteCommand`. After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetDeleteCommand` will still be using information from the previous statement.

**See also**

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

## GetInsertCommand method

Returns the generated `SACommand` object that performs INSERT operations on the database when an `Update` is called.

**Overload list**

Name	Description
<a href="#">GetInsertCommand() method</a>	Returns the generated <code>SACommand</code> object that performs INSERT operations on the database when an <code>Update</code> is called.
<a href="#">GetInsertCommand(bool) method</a>	Returns the generated <code>SACommand</code> object that performs INSERT operations on the database when an <code>Update</code> is called.

## GetInsertCommand() method

Returns the generated `SACommand` object that performs INSERT operations on the database when an `Update` is called.

**Visual Basic syntax**

```
Public Shadows Function GetInsertCommand() As SACommand
```

**C# syntax**

```
public new SACommand GetInsertCommand()
```

**Returns**

The automatically generated `SACommand` object required to perform insertions.

**Remarks**

The `GetInsertCommand` method returns the `SACommand` object to be executed, so it may be useful for informational or troubleshooting purposes.



You can also use `GetInsertCommand` as the basis of a modified command. For example, you might call `GetInsertCommand` and modify the `CommandTimeout` value, and then explicitly set that value on the `SADDataAdapter`.

SQL statements are first generated either when the application calls `Update` or `GetInsertCommand`. After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetInsertCommand` will be still be using information from the previous statement, which might not be correct.

### See also

- [“SACommandBuilder.GetDeleteCommand method \[SQL Anywhere .NET\]” on page 148](#)

## GetInsertCommand(bool) method

Returns the generated `SACommand` object that performs INSERT operations on the database when an `Update` is called.

### Visual Basic syntax

```
Public Shadows Function GetInsertCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

### C# syntax

```
public new SACommand GetInsertCommand(bool useColumnsForParameterNames)
```

### Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate `@p1`, `@p2`, and so on.

### Returns

The automatically generated `SACommand` object required to perform insertions.

### Remarks

The `GetInsertCommand` method returns the `SACommand` object to be executed, so it may be useful for informational or troubleshooting purposes.

You can also use `GetInsertCommand` as the basis of a modified command. For example, you might call `GetInsertCommand` and modify the `CommandTimeout` value, and then explicitly set that value on the `SADDataAdapter`.

SQL statements are first generated either when the application calls `Update` or `GetInsertCommand`. After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetInsertCommand` will be still be using information from the previous statement, which might not be correct.

### See also

- [“SACommandBuilder.GetDeleteCommand method \[SQL Anywhere .NET\]” on page 148](#)

## GetUpdateCommand method

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

### Overload list

Name	Description
<a href="#">GetUpdateCommand() method</a>	Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.
<a href="#">GetUpdateCommand(bool) method</a>	Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

### GetUpdateCommand() method

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

### Visual Basic syntax

```
Public Shadows Function GetUpdateCommand() As SACommand
```

### C# syntax

```
public new SACommand GetUpdateCommand()
```

### Returns

The automatically generated SACommand object required to perform updates.

### Remarks

The GetUpdateCommand method returns the SACommand object to be executed, so it may be useful for informational or troubleshooting purposes.

You can also use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that value on the SADataAdapter.

SQL statements are first generated when the application calls Update or GetUpdateCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand will be still be using information from the previous statement, which might not be correct.

### See also

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

## GetUpdateCommand(bool) method

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

### Visual Basic syntax

```
Public Shadows Function GetUpdateCommand(  
    ByVal useColumnsForParameterNames As Boolean  
) As SACommand
```

### C# syntax

```
public new SACommand GetUpdateCommand(bool useColumnsForParameterNames)
```

### Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

### Returns

The automatically generated SACommand object required to perform updates.

### Remarks

The GetUpdateCommand method returns the SACommand object to be executed, so it may be useful for informational or troubleshooting purposes.

You can also use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that value on the SADataAdapter.

SQL statements are first generated when the application calls Update or GetUpdateCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand will be still be using information from the previous statement, which might not be correct.

### See also

- [System.Data.Common.DbCommandBuilder.RefreshSchema](#)

## QuoteIdentifier method

Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

### Visual Basic syntax

```
Public Overrides Function QuoteIdentifier(  
    ByVal unquotedIdentifier As String  
) As String
```

### C# syntax

```
public override string QuoteIdentifier(string unquotedIdentifier)
```

### Parameters

- **unquotedIdentifier** The string representing the unquoted identifier that will have be quoted.

### Returns

Returns a string representing the quoted form of an unquoted identifier with embedded quotes properly escaped.

## UnquoteIdentifier method

Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.

### Visual Basic syntax

```
Public Overrides Function UnquoteIdentifier(  
    ByVal quotedIdentifier As String  
) As String
```

### C# syntax

```
public override string UnquoteIdentifier(string quotedIdentifier)
```

### Parameters

- **quotedIdentifier** The string representing the quoted identifier that will have its embedded quotes removed.

### Returns

Returns a string representing the unquoted form of a quoted identifier with embedded quotes properly un-escaped.

## DataAdapter property

Specifies the `SADDataAdapter` for which to generate statements.

### Visual Basic syntax

```
Public Shadows Property DataAdapter As SADDataAdapter
```

### C# syntax

```
public new SADDataAdapter DataAdapter {get;set;}
```

### Remarks

An `SADDataAdapter` object.

When you create a new instance of `SACCommandBuilder`, any existing `SACCommandBuilder` that is associated with this `SADDataAdapter` is released.

## SACCommLinksOptionsBuilder class

Provides a simple way to create and manage the `CommLinks` options portion of connection strings used by the `SACConnection` class.

### Visual Basic syntax

```
Public NotInheritable Class SACCommLinksOptionsBuilder
```

### C# syntax

```
public sealed class SACCommLinksOptionsBuilder
```

### Members

All members of `SACCommLinksOptionsBuilder` class, including all inherited members.

Name	Description
<a href="#">SACCommLinksOptionsBuilder constructor</a>	Initializes an <code>SACCommLinksOptionsBuilder</code> object.
<a href="#">GetUseLongNameAsKeyword method</a>	Gets a boolean values that indicates whether long connection parameter names are used in the connection string.
<a href="#">SetUseLongNameAsKeyword method</a>	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
<a href="#">ToString method</a>	Converts the <code>SACCommLinksOptionsBuilder</code> object to a string representation.
<a href="#">All property</a>	Gets or sets the ALL <code>CommLinks</code> option.
<a href="#">ConnectionString property</a>	Gets or sets the connection string being built.
<a href="#">SharedMemory property</a>	Gets or sets the <code>SharedMemory</code> protocol.
<a href="#">TcpOptionsBuilder property</a>	Gets or sets a <code>TcpOptionsBuilder</code> object used to create a TCP options string.
<a href="#">TcpOptionsString property</a>	Gets or sets a string of TCP options.

### Remarks

The `SACCommLinksOptionsBuilder` class is not available in the .NET Compact Framework 2.0.

For a list of connection parameters, see “[Connection parameters](#)” [*SQL Anywhere Server - Database Administration*].

## SACommLinksOptionsBuilder constructor

Initializes an SACommLinksOptionsBuilder object.

### Overload list

Name	Description
<a href="#">SACommLinksOptionsBuilder()</a> constructor	Initializes an SACommLinksOptionsBuilder object.
<a href="#">SACommLinksOptionsBuilder(string)</a> constructor	Initializes an SACommLinksOptionsBuilder object.

### SACommLinksOptionsBuilder() constructor

Initializes an SACommLinksOptionsBuilder object.

#### Visual Basic syntax

```
Public Sub New()
```

#### C# syntax

```
public SACommLinksOptionsBuilder()
```

#### Remarks

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

#### Example

The following statement initializes an SACommLinksOptionsBuilder object.

```
SACommLinksOptionsBuilder commLinks = new SACommLinksOptionsBuilder( );
```

### SACommLinksOptionsBuilder(string) constructor

Initializes an SACommLinksOptionsBuilder object.

#### Visual Basic syntax

```
Public Sub New(ByVal options As String)
```

#### C# syntax

```
public SACommLinksOptionsBuilder(string options)
```

**Parameters**

- **options** A SQL Anywhere CommLinks connection parameter string. For a list of connection parameters, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

**Remarks**

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

**Example**

The following statement initializes an SACommLinksOptionsBuilder object.

```
SACommLinksOptionsBuilder commLinks =  
    new SACommLinksOptionsBuilder("TCP/IP(DoBroadcast=ALL;Timeout=20)");
```

## GetUseLongNameAsKeyword method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

**Visual Basic syntax**

```
Public Function GetUseLongNameAsKeyword() As Boolean
```

**C# syntax**

```
public bool GetUseLongNameAsKeyword()
```

**Returns**

True if long connection parameter names are used to build connection strings; otherwise, false.

**Remarks**

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

**See also**

- [“SACommLinksOptionsBuilder.SetUseLongNameAsKeyword method \[SQL Anywhere .NET\]” on page 157](#)

## SetUseLongNameAsKeyword method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

### Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword(  
    ByVal useLongNameAsKeyword As Boolean  
)
```

### C# syntax

```
public void SetUseLongNameAsKeyword(bool useLongNameAsKeyword)
```

### Parameters

- **useLongNameAsKeyword** A boolean value that indicates whether the long connection parameter name is used in the connection string.

### Remarks

Long connection parameter names are used by default.

### See also

- [“SACommLinksOptionsBuilder.GetUseLongNameAsKeyword method \[SQL Anywhere .NET\]” on page 157](#)

## ToString method

Converts the SACommLinksOptionsBuilder object to a string representation.

### Visual Basic syntax

```
Public Overrides Function Tostring() As String
```

### C# syntax

```
public override string Tostring()
```

### Returns

The options string being built.

## All property

Gets or sets the ALL CommLinks option.

### Visual Basic syntax

```
Public Property All As Boolean
```

### C# syntax

```
public bool All {get;set;}
```



**Remarks**

Attempt to connect using the shared memory protocol first, followed by all remaining and available communication protocols. Use this setting if you are unsure of which communication protocol(s) to use.

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

## ConnectionString property

Gets or sets the connection string being built.

**Visual Basic syntax**

```
Public Property ConnectionString As String
```

**C# syntax**

```
public string ConnectionString {get;set;}
```

**Remarks**

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

## SharedMemory property

Gets or sets the SharedMemory protocol.

**Visual Basic syntax**

```
Public Property SharedMemory As Boolean
```

**C# syntax**

```
public bool SharedMemory {get;set;}
```

**Remarks**

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

## TcpOptionsBuilder property

Gets or sets a TcpOptionsBuilder object used to create a TCP options string.

**Visual Basic syntax**

```
Public Property TcpOptionsBuilder As SATcpOptionsBuilder
```

**C# syntax**

```
public SATcpOptionsBuilder TcpOptionsBuilder {get;set;}
```

## TcpOptionsString property

Gets or sets a string of TCP options.

### Visual Basic syntax

```
Public Property TcpOptionsString As String
```

### C# syntax

```
public string TcpOptionsString {get;set;}
```

## SACConnection class

Represents a connection to a SQL Anywhere database.

### Visual Basic syntax

```
Public NotInheritable Class SACConnection  
    Inherits System.Data.Common.DbConnection
```

### C# syntax

```
public sealed class SACConnection : System.Data.Common.DbConnection
```

### Base classes

- [System.Data.Common.DbConnection](#)

### Members

All members of SACConnection class, including all inherited members.

Name	Description
<a href="#">SACConnection constructor</a>	Initializes an SACConnection object.
<a href="#">BeginDbTransaction method</a> (Inherited from System.Data.Common.DbConnection)	Starts a database transaction.
<a href="#">BeginTransaction method</a>	Returns a transaction object.
<a href="#">ChangeDatabase method</a>	Changes the current database for an open SACConnection.
<a href="#">ChangePassword method</a>	Changes the password for the user indicated in the connection string to the supplied new password.
<a href="#">ClearAllPools method</a>	Empties all connection pools.

Name	Description
<a href="#">ClearPool method</a>	Empties the connection pool associated with the specified connection.
<a href="#">Close method</a>	Closes a database connection.
<a href="#">CreateCommand method</a>	Initializes an SACommand object.
<a href="#">EnlistDistributedTransaction method</a>	Enlists in the specified transaction as a distributed transaction.
<a href="#">EnlistTransaction method</a>	Enlists in the specified transaction as a distributed transaction.
<a href="#">GetSchema method</a>	Returns the list of supported schema collections.
<a href="#">OnStateChange method</a> (Inherited from System.Data.Common.DbConnection)	Raises the <a href="#">System.Data.Common.DbConnection.StateChange</a> event.
<a href="#">Open method</a>	Opens a database connection with the property settings specified by the <a href="#">SAConnection.ConnectionString</a> .
<a href="#">ConnectionString property</a>	Provides the database connection string.
<a href="#">ConnectionTimeout property</a>	Gets the number of seconds before a connection attempt times out with an error.
<a href="#">Database property</a>	Gets the name of the current database.
<a href="#">DataSource property</a>	Gets the name of the database server.
<a href="#">DbProviderFactory property</a> (Inherited from System.Data.Common.DbConnection)	Gets the <a href="#">System.Data.Common.DbProviderFactory</a> for this <a href="#">System.Data.Common.DbConnection</a> .
<a href="#">InitString property</a>	A command that is executed immediately after the connection is established.
<a href="#">ServerVersion property</a>	Gets a string that contains the version of the instance of SQL Anywhere to which the client is connected.
<a href="#">State property</a>	Indicates the state of the SAConnection object.
<a href="#">InfoMessage event</a>	Occurs when the SQL Anywhere database server returns a warning or informational message.
<a href="#">StateChange event</a>	Occurs when the state of the SAConnection object changes.

### Remarks

For a list of connection parameters, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

## SACConnection constructor

Initializes an SACConnection object.

### Overload list

Name	Description
<a href="#">SACConnection() constructor</a>	Initializes an SACConnection object.
<a href="#">SACConnection(string) constructor</a>	Initializes an SACConnection object.

### SACConnection() constructor

Initializes an SACConnection object.

#### Visual Basic syntax

```
Public Sub New()
```

#### C# syntax

```
public SACConnection()
```

### Remarks

The connection must be opened before you can perform any operations against the database.

### SACConnection(string) constructor

Initializes an SACConnection object.

#### Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

#### C# syntax

```
public SACConnection(string connectionString)
```

### Parameters

- **connectionString** A SQL Anywhere connection string. A connection string is a semicolon-separated list of keyword=value pairs. For a list of connection parameters, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

**Remarks**

The connection must then be opened before you can perform any operations against the database.

**See also**

- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

**Example**

The following statement initializes an SAConnection object for a connection to a database named policies running on a SQL Anywhere database server named hr. The connection uses the user ID admin and the password money.

```
SAConnection conn = new SAConnection(
    "UID=admin;PWD=money;SERVER=hr;DBN=policies" );
conn.Open();
```

## BeginTransaction method

Returns a transaction object.

**Overload list**

Name	Description
<a href="#">BeginTransaction() method</a>	Returns a transaction object.
<a href="#">BeginTransaction(IsolationLevel) method</a>	Returns a transaction object.
<a href="#">BeginTransaction(SAIsolationLevel) method</a>	Returns a transaction object.

### BeginTransaction() method

Returns a transaction object.

**Visual Basic syntax**

```
Public Shadows Function BeginTransaction() As SATransaction
```

**C# syntax**

```
public new SATransaction BeginTransaction()
```

**Returns**

An SATransaction object representing the new transaction.

**Remarks**

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the Commit or Rollback methods.

To associate a command with a transaction object, use the `SACCommand.Transaction` property.

#### See also

- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)
- [“SACCommand.Transaction property \[SQL Anywhere .NET\]” on page 143](#)

## BeginTransaction(IsolationLevel) method

Returns a transaction object.

#### Visual Basic syntax

```
Public Shadows Function BeginTransaction(  
    ByVal isolationLevel As IsolationLevel  
) As SATransaction
```

#### C# syntax

```
public new SATransaction BeginTransaction(IsolationLevel isolationLevel)
```

#### Parameters

- **isolationLevel** A member of the `SAIsolationLevel` enumeration. The default value is `ReadCommitted`.

#### Returns

An `SATransaction` object representing the new transaction.

#### Remarks

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the `Commit` or `Rollback` methods.

To associate a command with a transaction object, use the `SACCommand.Transaction` property.

#### See also

- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)
- [“SACCommand.Transaction property \[SQL Anywhere .NET\]” on page 143](#)
- [“SAIsolationLevel enumeration \[SQL Anywhere .NET\]” on page 324](#)

#### Example

```
SATransaction tx =  
    conn.BeginTransaction( SAIsolationLevel.ReadUncommitted );
```

## BeginTransaction(SAIsolationLevel) method

Returns a transaction object.

**Visual Basic syntax**

```
Public Function BeginTransaction(  
    ByVal isolationLevel As SAIsolationLevel  
) As SATransaction
```

**C# syntax**

```
public SATransaction BeginTransaction(SAIsolationLevel isolationLevel)
```

**Parameters**

- **isolationLevel** A member of the SAIsolationLevel enumeration. The default value is ReadCommitted.

**Returns**

An SATransaction object representing the new transaction.

**Remarks**

Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a call to the Commit or Rollback methods.

To associate a command with a transaction object, use the SACommand.Transaction property.

For more information, see [“Transaction processing” on page 62](#).

For more information, see [“Typical types of inconsistency” \[SQL Anywhere Server - SQL Usage\]](#).

**See also**

- [“SATransaction class \[SQL Anywhere .NET\]” on page 312](#)
- [“SACommand.Transaction property \[SQL Anywhere .NET\]” on page 143](#)
- [“SAIsolationLevel enumeration \[SQL Anywhere .NET\]” on page 324](#)
- [“SATransaction.Commit method \[SQL Anywhere .NET\]” on page 313](#)
- [“SATransaction.Rollback method \[SQL Anywhere .NET\]” on page 313](#)

## ChangeDatabase method

Changes the current database for an open SAConnection.

**Visual Basic syntax**

```
Public Overrides Sub ChangeDatabase(ByVal database As String)
```

**C# syntax**

```
public override void ChangeDatabase(string database)
```

**Parameters**

- **database** The name of the database to use instead of the current database.

## ChangePassword method

Changes the password for the user indicated in the connection string to the supplied new password.

### Visual Basic syntax

```
Public Shared Sub ChangePassword(  
    ByVal connectionString As String,  
    ByVal newPassword As String  
)
```

### C# syntax

```
public static void ChangePassword(  
    string connectionString,  
    string newPassword  
)
```

### Parameters

- **connectionString** The connection string that contains enough information to connect to the database server that you want. The connection string may contain the user ID and the current password.
- **newPassword** The new password to set. This password must comply with any password security policy set on the server, including minimum length, requirements for specific characters, and so on.

### Exceptions

- **ArgumentNullException** Either the *connectionString* or the *newPassword* parameter is null.
- **ArgumentException** The connection string includes the option to use integrated security.

## ClearAllPools method

Empties all connection pools.

### Visual Basic syntax

```
Public Shared Sub ClearAllPools()
```

### C# syntax

```
public static void ClearAllPools()
```

## ClearPool method

Empties the connection pool associated with the specified connection.

### Visual Basic syntax

```
Public Shared Sub ClearPool(ByVal connection As SAConnection)
```



**C# syntax**

```
public static void ClearPool(SAConnection connection)
```

**Parameters**

- **connection** The SAConnection object to be cleared from the pool.

**See also**

- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

## Close method

Closes a database connection.

**Visual Basic syntax**

```
Public Overrides Sub Close()
```

**C# syntax**

```
public override void Close()
```

**Remarks**

The Close method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled. If Close is called while handling a StateChange event, no additional StateChange events are fired. An application can call Close multiple times.

## CreateCommand method

Initializes an SACommand object.

**Visual Basic syntax**

```
Public Shadows Function CreateCommand() As SACommand
```

**C# syntax**

```
public new SACommand CreateCommand()
```

**Returns**

An SACommand object.

**Remarks**

The command object is associated with the SAConnection object.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)
- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

## EnlistDistributedTransaction method

Enlists in the specified transaction as a distributed transaction.

### Visual Basic syntax

```
Public Sub EnlistDistributedTransaction(  
    ByVal transaction As System.EnterpriseServices.ITransaction  
)
```

### C# syntax

```
public void EnlistDistributedTransaction(  
    System.EnterpriseServices.ITransaction transaction  
)
```

### Parameters

- **transaction** A reference to an existing System.EnterpriseServices.ITransaction in which to enlist.

## EnlistTransaction method

Enlists in the specified transaction as a distributed transaction.

### Visual Basic syntax

```
Public Overrides Sub EnlistTransaction(  
    ByVal transaction As System.Transactions.Transaction  
)
```

### C# syntax

```
public override void EnlistTransaction(  
    System.Transactions.Transaction transaction  
)
```

### Parameters

- **transaction** A reference to an existing System.Transactions.Transaction in which to enlist.

## GetSchema method

Returns the list of supported schema collections.

**Overload list**

Name	Description
<a href="#">GetSchema() method</a>	Returns the list of supported schema collections.
<a href="#">GetSchema(string) method</a>	Returns information for the specified metadata collection for this SAConnection object.
<a href="#">GetSchema(string, string[]) method</a>	Returns schema information for the data source of this SAConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

**GetSchema() method**

Returns the list of supported schema collections.

**Visual Basic syntax**

```
Public Overrides Function GetSchema() As DataTable
```

**C# syntax**

```
public override DataTable GetSchema()
```

**Remarks**

See [GetSchema\(string,string\[\]\)](#) for a description of the available metadata.

**GetSchema(string) method**

Returns information for the specified metadata collection for this SAConnection object.

**Visual Basic syntax**

```
Public Overrides Function GetSchema(  
    ByVal collection As String  
) As DataTable
```

**C# syntax**

```
public override DataTable GetSchema(string collection)
```

**Parameters**

- **collection** Name of the metadata collection. If a name is not provided, `MetaDataCollections` is used.

**Remarks**

See [GetSchema\(string,string\[\]\)](#) for a description of the available metadata.

**See also**

- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

**GetSchema(string, string[]) method**

Returns schema information for the data source of this SAConnection object and, if specified, uses the specified string for the schema name and the specified string array for the restriction values.

**Visual Basic syntax**

```
Public Overrides Function GetSchema(  
    ByVal collection As String,  
    ByVal restrictions As String()  
) As DataTable
```

**C# syntax**

```
public override DataTable GetSchema(  
    string collection,  
    string[] restrictions  
)
```

**Returns**

A DataTable that contains schema information.

**Remarks**

These methods are used to query the database server for various metadata. Each type of metadata is given a collection name, which must be passed to receive that data. The default collection name is `MetaDataCollections`.

You can query the SQL Anywhere SQL Anywhere .NET Data Provider to determine the list of supported schema collections by calling the `GetSchema` method with no arguments, or with the schema collection name **MetaDataCollections**. This will return a `DataTable` with a list of the supported schema collections (`CollectionName`), the number of restrictions that they each support (`NumberOfRestrictions`), and the number of identifier parts that they use (`NumberOfIdentifierParts`).

Collection	Metadata
Columns	Returns information on all columns in the database.
DataSourceInformation	Returns information about the database server.
DataTypes	Returns a list of supported data types.
ForeignKeys	Returns information on all foreign keys in the database.
IndexColumns	Returns information on all index columns in the database.
Indexes	Returns information on all indexes in the database.

Collection	Metadata
MetaDataCollections	Returns a list of all collection names.
ProcedureParameters	Returns information on all procedure parameters in the database.
Procedures	Returns information on all procedures in the database.
ReservedWords	Returns a list of reserved words used by SQL Anywhere.
Restrictions	Returns information on restrictions used in GetSchema.
Tables	Returns information on all tables in the database.
UserDefinedTypes	Returns information on all user-defined data types in the database.
Users	Returns information on all users in the database.
ViewColumns	Returns information on all columns in views in the database.
Views	Returns information on all views in the database.

These collection names are also available as read-only properties in the SAMetaDataCollectionNames class.

The results returned can be filtered by specifying an array of restrictions in the call to GetSchema.

The restrictions available with each collection can be queried by calling:

```
GetSchema( "Restrictions" )
```

If the collection requires four restrictions, then the restrictions parameter must be either NULL, or a string with four values.

To filter on a particular restriction, place the string to filter by in its place in the array and leave any unused places NULL. For example, the Tables collection has three restrictions: Owner, Table, and TableType.

To filter the Table collection by table\_name:

```
GetSchema( "Tables", new string[ ] { NULL, "my_table", NULL } )
```

This returns information on all tables named my\_table.

```
GetSchema( "Tables", new string[ ] { "DBA", "my_table", NULL } )
```

This returns information on all tables named my\_table owned by the user DBA.

The following is a summary of the columns returned by each collection. If the number of rows returned in a collection can be reduced by specifying a restriction on a column, the restriction name for that column is shown in parenthesis. The order in which restrictions are specified is the order in which they are presented in the lists below.

**Columns** collection

- table\_schema (Owner)
- table\_name (Table)
- column\_name (Column)
- ordinal\_position
- column\_default
- is\_nullable
- data\_type
- precision
- scale
- column\_size

**DataSourceInformation** collection

- CompositeIdentifierSeparatorPattern
- DataSourceProductName
- DataSourceProductVersion
- DataSourceProductVersionNormalized
- GroupByBehavior
- IdentifierPattern
- IdentifierCase
- OrderByColumnsInSelect
- ParameterMarkerFormat
- ParameterMarkerPattern
- ParameterNameMaxLength
- ParameterNamePattern
- QuotedIdentifierPattern
- QuotedIdentifierCase
- StatementSeparatorPattern

- StringLiteralPattern
- SupportedJoinOperators

**DataTypes** collection

- TypeName
- ProviderDbType
- ColumnSize
- CreateFormat
- CreateParameters
- DataType
- IsAutoIncrementable
- IsBestMatch
- IsCaseSensitive
- IsFixedLength
- IsFixedPrecisionScale
- IsLong
- IsNullable
- IsSearchable
- IsSearchableWithLike
- IsUnsigned
- MaximumScale
- MinimumScale
- IsConcurrencyType
- IsLiteralSupported
- LiteralPrefix
- LiteralSuffix

**ForeignKeys** collection

- table\_schema (Owner)

- table\_name (Table)
- column\_name (Column)

**IndexColumns** collection

- table\_schema (Owner)
- table\_name (Table)
- index\_name (Name)
- column\_name (Column)
- order

**Indexes** collection

- table\_schema (Owner)
- table\_name (Table)
- index\_name (Name)
- primary\_key
- is\_unique

**MetaDataCollections** collection

- CollectionName
- NumberOfRestrictions
- NumberOfIdentifierParts

**ProcedureParameters** collection

- procedure\_schema (Owner)
- procedure\_name (Name)
- parameter\_name (Parameter)
- data\_type
- parameter\_type
- is\_input
- is\_output

**Procedures** collection



- procedure\_schema (Owner)
- procedure\_name (Name)

**ReservedWords** collection

- reserved\_word

**Restrictions** collection

- CollectionName
- RestrictionName
- RestrictionDefault
- RestrictionNumber

**Tables** collection

- table\_schema (Owner)
- table\_name (Table)
- table\_type (TableType) either "BASE", "VIEW", "MAT VIEW", "LCL TEMP", "GBL TEMP", "TEXT", or "TEXT GBL TEMP"

**UserDefinedTypes** collection

- data\_type
- default
- precision
- scale

**Users** collection

- user\_name (UserName)
- resource\_auth
- database\_auth
- schedule\_auth
- user\_group

**ViewColumns** collection

- view\_schema (Owner)

- view\_name (Name)
- column\_name (Column)

**Views** collection

- view\_schema (Owner)
- view\_name (Name)

**See also**

- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

## Open method

Opens a database connection with the property settings specified by the SAConnection.ConnectionString.

**Visual Basic syntax**

```
Public Overrides Sub Open()
```

**C# syntax**

```
public override void Open()
```

**See also**

- [“SAConnection.ConnectionString property \[SQL Anywhere .NET\]” on page 176](#)

## ConnectionString property

Provides the database connection string.

**Visual Basic syntax**

```
Public Overrides Property ConnectionString As String
```

**C# syntax**

```
public override string ConnectionString {get;set;}
```

**Remarks**

The ConnectionString is designed to match the SQL Anywhere connection string format as closely as possible with the following exception: when the Persist Security Info value is set to false (the default), the connection string that is returned is the same as the user-set ConnectionString minus security information. The SQL Anywhere SQL Anywhere .NET Data Provider does not persist the password in a returned connection string unless you set Persist Security Info to true.

You can use the ConnectionString property to connect to a variety of data sources.

You can set the `ConnectionString` property only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, unless an error is detected. If an error is detected, none of the properties are updated. `SACConnection` properties return only those settings contained in the `ConnectionString`.

If you reset the `ConnectionString` on a closed connection, all connection string values and related properties are reset, including the password.

When the property is set, a preliminary validation of the connection string is performed. When an application calls the `Open` method, the connection string is fully validated. A runtime exception is generated if the connection string contains invalid or unsupported properties.

Values can be delimited by single or double quotes. Either single or double quotes may be used within a connection string by using the other delimiter, for example, `name="value's"` or `name='value"s'`, but not `name='value's'` or `name= ""value""`. Blank characters are ignored unless they are placed within a value or within quotes. `keyword=value` pairs must be separated by a semicolon. If a semicolon is part of a value, it must also be delimited by quotes. Escape sequences are not supported, and the value type is irrelevant. Names are not case sensitive. If a property name occurs more than once in the connection string, the value associated with the last occurrence is used.

You should use caution when constructing a connection string based on user input, such as when retrieving a user ID and password from a window, and appending it to the connection string. The application should not allow a user to embed extra connection string parameters in these values.

The default value of connection pooling is `true` (`pooling=true`).

### See also

- [“SACConnection class \[SQL Anywhere .NET\]” on page 160](#)
- [“SACConnection.Open method \[SQL Anywhere .NET\]” on page 176](#)

### Example

The following statements set a connection string for an ODBC data source named SQL Anywhere 12 Demo and open the connection.

```
SACConnection conn = new SACConnection();
conn.ConnectionString = "DSN=SQL Anywhere 12 Demo";
conn.Open();
```

## ConnectionTimeout property

Gets the number of seconds before a connection attempt times out with an error.

### Visual Basic syntax

```
Public ReadOnly Overrides Property ConnectionTimeout As Integer
```

### C# syntax

```
public override int ConnectionTimeout {get;}
```

### Remarks

15 seconds

### Example

The following statement displays the value of the ConnectionTimeout.

```
MessageBox.Show( conn.ConnectionTimeout.ToString( ) );
```

## Database property

Gets the name of the current database.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Database As String
```

### C# syntax

```
public override string Database {get;}
```

### Remarks

If the connection is opened, SACConnection returns the name of the current database. Otherwise, SACConnection looks in the connection string in the following order: DatabaseName, DBN, DataSourceName, DataSource, DSN, DatabaseFile, DBF.

## DataSource property

Gets the name of the database server.

### Visual Basic syntax

```
Public ReadOnly Overrides Property DataSource As String
```

### C# syntax

```
public override string DataSource {get;}
```

### Remarks

If the connection is opened, the SACConnection object returns the ServerName server property. Otherwise, the SACConnection object looks in the connection string in the following order: EngineName, ServerName, Server, ENG.

### See also

- [“SACConnection class \[SQL Anywhere .NET\]” on page 160](#)

## InitString property

A command that is executed immediately after the connection is established.

### Visual Basic syntax

```
Public Property InitString As String
```

### C# syntax

```
public string InitString {get;set;}
```

### Remarks

The InitString will be executed immediately after the connection is opened.

## ServerVersion property

Gets a string that contains the version of the instance of SQL Anywhere to which the client is connected.

### Visual Basic syntax

```
Public ReadOnly Overrides Property ServerVersion As String
```

### C# syntax

```
public override string ServerVersion {get;}
```

### Returns

The version of the instance of SQL Anywhere.

### Remarks

The version is `##.##.####`, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The appended string is of the form `major.minor.build`, where `major` and `minor` are two digits, and `build` is four digits.

## State property

Indicates the state of the SACConnection object.

### Visual Basic syntax

```
Public ReadOnly Overrides Property State As ConnectionState
```

### C# syntax

```
public override ConnectionState State {get;}
```

### Returns

A System.Data.ConnectionState enumeration.

## InfoMessage event

Occurs when the SQL Anywhere database server returns a warning or informational message.

### Visual Basic syntax

```
Public Event InfoMessage As SAInfoMessageEventHandler
```

### C# syntax

```
public event SAInfoMessageEventHandler InfoMessage;
```

### Remarks

The event handler receives an argument of type `SAInfoMessageEventArgs` containing data related to this event. The following `SAInfoMessageEventArgs` properties provide information specific to this event: `NativeError`, `Errors`, `Message`, `MessageType`, and `Source`.

For more information, see the .NET Framework documentation for `OleDbConnection.InfoMessage` Event.

## StateChange event

Occurs when the state of the `SACConnection` object changes.

### Visual Basic syntax

```
Public Event StateChange As StateChangeEventHandler
```

### C# syntax

```
public event override StateChangeEventHandler StateChange;
```

### Remarks

The event handler receives an argument of type `StateChangeEventArgs` with data related to this event. The following `StateChangeEventArgs` properties provide information specific to this event: `CurrentState` and `OriginalState`.

For more information, see the .NET Framework documentation for `OleDbConnection.StateChange` Event.

## SACConnectionStringBuilder class

Provides a simple way to create and manage the contents of connection strings used by the `SACConnection` class.

### Visual Basic syntax

```
Public NotInheritable Class SACConnectionStringBuilder  
    Inherits SACConnectionStringBuilderBase
```

**C# syntax**

```
public sealed class SAConnectionStringBuilder :
    SAConnectionStringBuilderBase
```

**Base classes**

- [“SAConnectionStringBuilderBase class \[SQL Anywhere .NET\]” on page 198](#)

**Members**

All members of SAConnectionStringBuilder class, including all inherited members.

Name	Description
<a href="#">SAConnectionStringBuilder constructor</a>	Initializes a new instance of the SAConnectionStringBuilder class.
<a href="#">Add method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Adds an entry with the specified key and value into the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">AppendKeyValuePair method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Provides an efficient and safe way to append a key and value to an existing <a href="#">System.Text.StringBuilder</a> object.
<a href="#">Clear method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the contents of the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> instance.
<a href="#">ClearPropertyDescriptors method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the collection of <a href="#">System.ComponentModel.PropertyDescriptor</a> objects on the associated <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">ContainsKey method</a>	Determines whether the SAConnectionStringBuilder object contains a specific keyword.
<a href="#">EquivalentTo method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Compares the connection information in this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> object with the connection information in the supplied object.
<a href="#">GetKeyword method</a>	Gets the keyword for specified SAConnectionStringBuilder property.
<a href="#">GetProperties method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Fills a supplied <a href="#">System.Collections.Hashtable</a> with information about all the properties of this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">GetUseLongNameAsKeyword method</a>	Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

Name	Description
<a href="#">Remove method</a>	Removes the entry with the specified key from the <code>SACConnectionStringBuilder</code> instance.
<a href="#">SetUseLongNameAsKeyword method</a>	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
<a href="#">ShouldSerialize method</a>	Indicates whether the specified key exists in this <code>SACConnectionStringBuilder</code> instance.
<a href="#">ToString method</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Returns the connection string associated with this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">TryGetValue method</a>	Retrieves a value corresponding to the supplied key from this <code>SACConnectionStringBuilder</code> .
<a href="#">AppInfo property</a>	Gets or sets the <code>AppInfo</code> connection property.
<a href="#">AutoStart property</a>	Gets or sets the <code>AutoStart</code> connection property.
<a href="#">AutoStop property</a>	Gets or sets the <code>AutoStop</code> connection property.
<a href="#">BrowsableConnectionString property</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Gets or sets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder.ConnectionString</a> property is visible in Visual Studio designers.
<a href="#">Charset property</a>	Gets or sets the <code>Charset</code> connection property.
<a href="#">CommBufferSize property</a>	Gets or sets the <code>CommBufferSize</code> connection property.
<a href="#">CommLinks property</a>	Gets or sets the <code>CommLinks</code> property.
<a href="#">Compress property</a>	Gets or sets the <code>Compress</code> connection property.
<a href="#">CompressionThreshold property</a>	Gets or sets the <code>CompressionThreshold</code> connection property.
<a href="#">ConnectionLifetime property</a>	Gets or sets the <code>ConnectionLifetime</code> connection property.
<a href="#">ConnectionName property</a>	Gets or sets the <code>ConnectionName</code> connection property.
<a href="#">ConnectionPool property</a>	Gets or sets the <code>ConnectionPool</code> property.
<a href="#">ConnectionReset property</a>	Gets or sets the <code>ConnectionReset</code> connection property.



Name	Description
<a href="#">ConnectionString</a> property (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets or sets the connection string associated with the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">ConnectionTimeout</a> property	Gets or sets the <a href="#">ConnectionTimeout</a> connection property.
<a href="#">Count</a> property (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets the current number of keys that are contained within the <a href="#">System.Data.Common.DbConnectionStringBuilder.ConnectionString</a> property.
<a href="#">DatabaseFile</a> property	Gets or sets the <a href="#">DatabaseFile</a> connection property.
<a href="#">DatabaseKey</a> property	Gets or sets the <a href="#">DatabaseKey</a> connection property.
<a href="#">DatabaseName</a> property	Gets or sets the <a href="#">DatabaseName</a> connection property.
<a href="#">DatabaseSwitches</a> property	Gets or sets the <a href="#">DatabaseSwitches</a> connection property.
<a href="#">DataSourceName</a> property	Gets or sets the <a href="#">DataSourceName</a> connection property.
<a href="#">DisableMultiRowFetch</a> property	Gets or sets the <a href="#">DisableMultiRowFetch</a> connection property.
<a href="#">Elevate</a> property	Gets or sets the <a href="#">Elevate</a> connection property.
<a href="#">EncryptedPassword</a> property	Gets or sets the <a href="#">EncryptedPassword</a> connection property.
<a href="#">Encryption</a> property	Gets or sets the <a href="#">Encryption</a> connection property.
<a href="#">Enlist</a> property	Gets or sets the <a href="#">Enlist</a> connection property.
<a href="#">FileDataSourceName</a> property	Gets or sets the <a href="#">FileDataSourceName</a> connection property.
<a href="#">ForceStart</a> property	Gets or sets the <a href="#">ForceStart</a> connection property.
<a href="#">Host</a> property	Gets or sets the <a href="#">Host</a> property.
<a href="#">IdleTimeout</a> property	Gets or sets the <a href="#">IdleTimeout</a> connection property.
<a href="#">InitString</a> property	Gets or sets the <a href="#">InitString</a> connection property.
<a href="#">Integrated</a> property	Gets or sets the <a href="#">Integrated</a> connection property.
<a href="#">IsFixedSize</a> property (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> has a fixed size.

Name	Description
<a href="#">IsReadOnly property</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> is read-only.
<a href="#">Kerberos property</a>	Gets or sets the Kerberos connection property.
<a href="#">Keys property</a>	Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.
<a href="#">Language property</a>	Gets or sets the Language connection property.
<a href="#">LazyClose property</a>	Gets or sets the LazyClose connection property.
<a href="#">LivenessTimeout property</a>	Gets or sets the LivenessTimeout connection property.
<a href="#">LogFile property</a>	Gets or sets the LogFile connection property.
<a href="#">MaxPoolSize property</a>	Gets or sets the MaxPoolSize connection property.
<a href="#">MinPoolSize property</a>	Gets or sets the MinPoolSize connection property.
<a href="#">NewPassword property</a>	Gets or sets the NewPassword connection property.
<a href="#">NodeType property</a>	Gets or sets the NodeType property.
<a href="#">Password property</a>	Gets or sets the Password connection property.
<a href="#">PersistSecurityInfo property</a>	Gets or sets the PersistSecurityInfo connection property.
<a href="#">Pooling property</a>	Gets or sets the Pooling connection property.
<a href="#">PrefetchBuffer property</a>	Gets or sets the PrefetchBuffer connection property.
<a href="#">PrefetchRows property</a>	Gets or sets the PrefetchRows connection property.
<a href="#">RetryConnectionTimeout property</a>	Gets or sets the RetryConnectionTimeout property.
<a href="#">ServerName property</a>	Gets or sets the ServerName connection property.
<a href="#">StartLine property</a>	Gets or sets the StartLine connection property.
<a href="#">this property</a>	Gets or sets the value of the connection keyword.
<a href="#">Unconditional property</a>	Gets or sets the Unconditional connection property.
<a href="#">UserID property</a>	Gets or sets the UserID connection property.

Name	Description
<a href="#">Values property</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets an <a href="#">System.Collections.ICollection</a> that contains the values in the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .

**Remarks**

The SAConnectionStringBuilder class inherits SAConnectionStringBuilderBase, which inherits DbConnectionStringBuilder.

The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

For a list of connection parameters, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

**SAConnectionStringBuilder constructor**

Initializes a new instance of the SAConnectionStringBuilder class.

**Overload list**

Name	Description
<a href="#">SAConnectionStringBuilder() constructor</a>	Initializes a new instance of the SAConnectionStringBuilder class.
<a href="#">SAConnectionStringBuilder(string) constructor</a>	Initializes a new instance of the SAConnectionStringBuilder class.

**SAConnectionStringBuilder() constructor**

Initializes a new instance of the SAConnectionStringBuilder class.

**Visual Basic syntax**

```
Public Sub New()
```

**C# syntax**

```
public SAConnectionStringBuilder()
```

**Remarks**

The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

**SAConnectionStringBuilder(string) constructor**

Initializes a new instance of the SAConnectionStringBuilder class.

### Visual Basic syntax

```
Public Sub New(ByVal connectionString As String)
```

### C# syntax

```
public SAConnectionStringBuilder(string connectionString)
```

### Parameters

- **connectionString** The basis for the object's internal connection information. Parsed into keyword=value pairs. For a list of connection parameters, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

### Remarks

The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

### Example

The following statement initializes an SAConnection object for a connection to a database named policies running on a SQL Anywhere database server named hr. The connection uses the user ID admin and the password money.

```
SAConnectionStringBuilder conn = new SAConnectionStringBuilder(  
    "UID=admin;PWD=money;SERVER=hr;DBN=policies" );
```

## AppInfo property

Gets or sets the AppInfo connection property.

### Visual Basic syntax

```
Public Property AppInfo As String
```

### C# syntax

```
public string AppInfo {get;set;}
```

## AutoStart property

Gets or sets the AutoStart connection property.

### Visual Basic syntax

```
Public Property AutoStart As String
```

### C# syntax

```
public string AutoStart {get;set;}
```

## AutoStop property

Gets or sets the AutoStop connection property.

### Visual Basic syntax

```
Public Property AutoStop As String
```

### C# syntax

```
public string AutoStop {get;set;}
```

## Charset property

Gets or sets the Charset connection property.

### Visual Basic syntax

```
Public Property Charset As String
```

### C# syntax

```
public string Charset {get;set;}
```

## CommBufferSize property

Gets or sets the CommBufferSize connection property.

### Visual Basic syntax

```
Public Property CommBufferSize As Integer
```

### C# syntax

```
public int CommBufferSize {get;set;}
```

## CommLinks property

Gets or sets the CommLinks property.

### Visual Basic syntax

```
Public Property CommLinks As String
```

### C# syntax

```
public string CommLinks {get;set;}
```

## Compress property

Gets or sets the Compress connection property.

### Visual Basic syntax

```
Public Property Compress As String
```

### C# syntax

```
public string Compress {get;set;}
```

## CompressionThreshold property

Gets or sets the CompressionThreshold connection property.

### Visual Basic syntax

```
Public Property CompressionThreshold As Integer
```

### C# syntax

```
public int CompressionThreshold {get;set;}
```

## ConnectionLifetime property

Gets or sets the ConnectionLifetime connection property.

### Visual Basic syntax

```
Public Property ConnectionLifetime As Integer
```

### C# syntax

```
public int ConnectionLifetime {get;set;}
```

## ConnectionString property

Gets or sets the ConnectionName connection property.

### Visual Basic syntax

```
Public Property ConnectionString As String
```

### C# syntax

```
public string ConnectionString {get;set;}
```

## ConnectionPool property

Gets or sets the ConnectionPool property.

### Visual Basic syntax

```
Public Property ConnectionPool As String
```

### C# syntax

```
public string ConnectionPool {get;set;}
```

## ConnectionReset property

Gets or sets the ConnectionReset connection property.

### Visual Basic syntax

```
Public Property ConnectionReset As Boolean
```

### C# syntax

```
public bool ConnectionReset {get;set;}
```

### Returns

A DataTable that contains schema information.

## ConnectionTimeout property

Gets or sets the ConnectionTimeout connection property.

### Visual Basic syntax

```
Public Property ConnectionTimeout As Integer
```

### C# syntax

```
public int ConnectionTimeout {get;set;}
```

### Example

The following statement displays the value of the ConnectionTimeout property.

```
MessageBox.Show( connString.ConnectionTimeout.ToString() );
```

## DatabaseFile property

Gets or sets the DatabaseFile connection property.

### Visual Basic syntax

```
Public Property DatabaseFile As String
```

### C# syntax

```
public string DatabaseFile {get;set;}
```

## DatabaseKey property

Gets or sets the DatabaseKey connection property.

### Visual Basic syntax

```
Public Property DatabaseKey As String
```

### C# syntax

```
public string DatabaseKey {get;set;}
```

## DatabaseName property

Gets or sets the DatabaseName connection property.

### Visual Basic syntax

```
Public Property DatabaseName As String
```

### C# syntax

```
public string DatabaseName {get;set;}
```

## DatabaseSwitches property

Gets or sets the DatabaseSwitches connection property.

### Visual Basic syntax

```
Public Property DatabaseSwitches As String
```

### C# syntax

```
public string DatabaseSwitches {get;set;}
```

## DataSourceName property

Gets or sets the DataSourceName connection property.



**Visual Basic syntax**

```
Public Property DataSourceName As String
```

**C# syntax**

```
public string DataSourceName {get;set;}
```

## DisableMultiRowFetch property

Gets or sets the DisableMultiRowFetch connection property.

**Visual Basic syntax**

```
Public Property DisableMultiRowFetch As String
```

**C# syntax**

```
public string DisableMultiRowFetch {get;set;}
```

## Elevate property

Gets or sets the Elevate connection property.

**Visual Basic syntax**

```
Public Property Elevate As String
```

**C# syntax**

```
public string Elevate {get;set;}
```

## EncryptedPassword property

Gets or sets the EncryptedPassword connection property.

**Visual Basic syntax**

```
Public Property EncryptedPassword As String
```

**C# syntax**

```
public string EncryptedPassword {get;set;}
```

## Encryption property

Gets or sets the Encryption connection property.

### Visual Basic syntax

```
Public Property Encryption As String
```

### C# syntax

```
public string Encryption {get;set;}
```

## Enlist property

Gets or sets the Enlist connection property.

### Visual Basic syntax

```
Public Property Enlist As Boolean
```

### C# syntax

```
public bool Enlist {get;set;}
```

## FileDataSourceName property

Gets or sets the FileDataSourceName connection property.

### Visual Basic syntax

```
Public Property FileDataSourceName As String
```

### C# syntax

```
public string FileDataSourceName {get;set;}
```

## ForceStart property

Gets or sets the ForceStart connection property.

### Visual Basic syntax

```
Public Property ForceStart As String
```

### C# syntax

```
public string ForceStart {get;set;}
```

## Host property

Gets or sets the Host property.

**Visual Basic syntax**

```
Public Property Host As String
```

**C# syntax**

```
public string Host {get;set;}
```

## IdleTimeout property

Gets or sets the IdleTimeout connection property.

**Visual Basic syntax**

```
Public Property IdleTimeout As Integer
```

**C# syntax**

```
public int IdleTimeout {get;set;}
```

## InitString property

Gets or sets the InitString connection property.

**Visual Basic syntax**

```
Public Property InitString As String
```

**C# syntax**

```
public string InitString {get;set;}
```

## Integrated property

Gets or sets the Integrated connection property.

**Visual Basic syntax**

```
Public Property Integrated As String
```

**C# syntax**

```
public string Integrated {get;set;}
```

## Kerberos property

Gets or sets the Kerberos connection property.

### Visual Basic syntax

```
Public Property Kerberos As String
```

### C# syntax

```
public string Kerberos {get;set;}
```

## Language property

Gets or sets the Language connection property.

### Visual Basic syntax

```
Public Property Language As String
```

### C# syntax

```
public string Language {get;set;}
```

## LazyClose property

Gets or sets the LazyClose connection property.

### Visual Basic syntax

```
Public Property LazyClose As String
```

### C# syntax

```
public string LazyClose {get;set;}
```

## LivenessTimeout property

Gets or sets the LivenessTimeout connection property.

### Visual Basic syntax

```
Public Property LivenessTimeout As Integer
```

### C# syntax

```
public int LivenessTimeout {get;set;}
```

## LogFile property

Gets or sets the LogFile connection property.

**Visual Basic syntax**

```
Public Property LogFile As String
```

**C# syntax**

```
public string LogFile {get;set;}
```

## MaxPoolSize property

Gets or sets the MaxPoolSize connection property.

**Visual Basic syntax**

```
Public Property MaxPoolSize As Integer
```

**C# syntax**

```
public int MaxPoolSize {get;set;}
```

## MinPoolSize property

Gets or sets the MinPoolSize connection property.

**Visual Basic syntax**

```
Public Property MinPoolSize As Integer
```

**C# syntax**

```
public int MinPoolSize {get;set;}
```

## NewPassword property

Gets or sets the NewPassword connection property.

**Visual Basic syntax**

```
Public Property NewPassword As String
```

**C# syntax**

```
public string NewPassword {get;set;}
```

## NodeType property

Gets or sets the NodeType property.

### Visual Basic syntax

```
Public Property NodeType As String
```

### C# syntax

```
public string NodeType {get;set;}
```

## Password property

Gets or sets the Password connection property.

### Visual Basic syntax

```
Public Property Password As String
```

### C# syntax

```
public string Password {get;set;}
```

## PersistSecurityInfo property

Gets or sets the PersistSecurityInfo connection property.

### Visual Basic syntax

```
Public Property PersistSecurityInfo As Boolean
```

### C# syntax

```
public bool PersistSecurityInfo {get;set;}
```

## Pooling property

Gets or sets the Pooling connection property.

### Visual Basic syntax

```
Public Property Pooling As Boolean
```

### C# syntax

```
public bool Pooling {get;set;}
```

## PrefetchBuffer property

Gets or sets the PrefetchBuffer connection property.

**Visual Basic syntax**

```
Public Property PrefetchBuffer As Integer
```

**C# syntax**

```
public int PrefetchBuffer {get;set;}
```

## PrefetchRows property

Gets or sets the PrefetchRows connection property.

**Visual Basic syntax**

```
Public Property PrefetchRows As Integer
```

**C# syntax**

```
public int PrefetchRows {get;set;}
```

**Remarks**

The default value is 200.

## RetryConnectionTimeout property

Gets or sets the RetryConnectionTimeout property.

**Visual Basic syntax**

```
Public Property RetryConnectionTimeout As Integer
```

**C# syntax**

```
public int RetryConnectionTimeout {get;set;}
```

## ServerName property

Gets or sets the ServerName connection property.

**Visual Basic syntax**

```
Public Property ServerName As String
```

**C# syntax**

```
public string ServerName {get;set;}
```

## StartLine property

Gets or sets the StartLine connection property.

### Visual Basic syntax

```
Public Property StartLine As String
```

### C# syntax

```
public string StartLine {get;set;}
```

## Unconditional property

Gets or sets the Unconditional connection property.

### Visual Basic syntax

```
Public Property Unconditional As String
```

### C# syntax

```
public string Unconditional {get;set;}
```

## UserID property

Gets or sets the UserID connection property.

### Visual Basic syntax

```
Public Property UserID As String
```

### C# syntax

```
public string UserID {get;set;}
```

## SACConnectionStringBuilderBase class

Base class of the SACConnectionStringBuilder class.

### Visual Basic syntax

```
Public MustInherit Class SACConnectionStringBuilderBase  
    Inherits System.Data.Common.DbConnectionStringBuilder
```

### C# syntax

```
public abstract class SACConnectionStringBuilderBase :  
    System.Data.Common.DbConnectionStringBuilder
```



**Base classes**

- [System.Data.Common.DbConnectionStringBuilder](#)

**Derived classes**

- “[SAConnectionStringBuilder class \[SQL Anywhere .NET\]](#)” on page 180
- “[SATcpOptionsBuilder class \[SQL Anywhere .NET\]](#)” on page 304

**Members**

All members of [SAConnectionStringBuilderBase](#) class, including all inherited members.

Name	Description
<a href="#">Add method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Adds an entry with the specified key and value into the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">AppendKeyValuePair method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Provides an efficient and safe way to append a key and value to an existing <a href="#">System.Text.StringBuilder</a> object.
<a href="#">Clear method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Clears the contents of the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> instance.
<a href="#">ClearPropertyDescriptors method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Clears the collection of <a href="#">System.ComponentModel.PropertyDescriptor</a> objects on the associated <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">ContainsKey method</a>	Determines whether the <a href="#">SAConnectionStringBuilder</a> object contains a specific keyword.
<a href="#">EquivalentTo method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Compares the connection information in this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> object with the connection information in the supplied object.
<a href="#">GetKeyword method</a>	Gets the keyword for specified <a href="#">SAConnectionStringBuilder</a> property.
<a href="#">GetProperties method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Fills a supplied <a href="#">System.Collections.Hashtable</a> with information about all the properties of this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">GetUseLongNameAsKeyword method</a>	Gets a boolean values that indicates whether long connection parameter names are used in the connection string.
<a href="#">Remove method</a>	Removes the entry with the specified key from the <a href="#">SAConnectionStringBuilder</a> instance.

Name	Description
<a href="#">SetUseLongNameAsKeyword method</a>	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
<a href="#">ShouldSerialize method</a>	Indicates whether the specified key exists in this <a href="#">SAConnectionStringBuilder</a> instance.
<a href="#">ToString method</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Returns the connection string associated with this <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">TryGetValue method</a>	Retrieves a value corresponding to the supplied key from this <a href="#">SAConnectionStringBuilder</a> .
<a href="#">BrowsableConnectionString property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets or sets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder.ConnectionString</a> property is visible in Visual Studio designers.
<a href="#">ConnectionString property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets or sets the connection string associated with the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">Count property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets the current number of keys that are contained within the <a href="#">System.Data.Common.DbConnectionStringBuilder.ConnectionString</a> property.
<a href="#">IsFixedSize property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> has a fixed size.
<a href="#">IsReadOnly property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> is read-only.
<a href="#">Keys property</a>	Gets an <a href="#">System.Collections.ICollection</a> that contains the keys in the <a href="#">SAConnectionStringBuilder</a> .
<a href="#">this property</a>	Gets or sets the value of the connection keyword.
<a href="#">Values property</a> (Inherited from <a href="#">System.Data.Common.DbConnectionStringBuilder</a> )	Gets an <a href="#">System.Collections.ICollection</a> that contains the values in the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .

## ContainsKey method

Determines whether the [SAConnectionStringBuilder](#) object contains a specific keyword.

**Visual Basic syntax**

```
Public Overrides Function ContainsKey(  
    ByVal keyword As String  
) As Boolean
```

**C# syntax**

```
public override bool ContainsKey(string keyword)
```

**Parameters**

- **keyword** The keyword to locate in the SAConnectionStringBuilder.

**Returns**

True if the value associated with keyword has been set; otherwise, false.

**Example**

The following statement determines whether the SAConnectionStringBuilder object contains the UserID keyword.

```
connectString.ContainsKey("UserID")
```

## GetKeyword method

Gets the keyword for specified SAConnectionStringBuilder property.

**Visual Basic syntax**

```
Public Function GetKeyword(ByVal propName As String) As String
```

**C# syntax**

```
public string GetKeyword(string propName)
```

**Parameters**

- **propName** The name of the SAConnectionStringBuilder property.

**Returns**

The keyword for specified SAConnectionStringBuilder property.

## GetUseLongNameAsKeyword method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

**Visual Basic syntax**

```
Public Function GetUseLongNameAsKeyword() As Boolean
```

### C# syntax

```
public bool GetUseLongNameAsKeyword( )
```

### Returns

True if long connection parameter names are used to build connection strings; otherwise, false.

### Remarks

SQL Anywhere connection parameters have both long and short forms of their names. For example, to specify the name of an ODBC data source in your connection string, you can use either of the following values: DataSourceName or DSN. By default, long connection parameter names are used to build connection strings.

### See also

- [“SAConnectionStringBuilderBase.SetUseLongNameAsKeyword method \[SQL Anywhere .NET\]” on page 202](#)

## Remove method

Removes the entry with the specified key from the SAConnectionStringBuilder instance.

### Visual Basic syntax

```
Public Overrides Function Remove(ByVal keyword As String) As Boolean
```

### C# syntax

```
public override bool Remove(string keyword)
```

### Parameters

- **keyword** The key of the key/value pair to be removed from the connection string in this SAConnectionStringBuilder.

### Returns

True if the key existed within the connection string and was removed; false if the key did not exist.

## SetUseLongNameAsKeyword method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string.

### Visual Basic syntax

```
Public Sub SetUseLongNameAsKeyword(  
    ByVal useLongNameAsKeyword As Boolean  
)
```

**C# syntax**

```
public void SetUseLongNameAsKeyword(bool useLongNameAsKeyword)
```

**Parameters**

- **useLongNameAsKeyword** A boolean value that indicates whether the long connection parameter name is used in the connection string.

**Remarks**

Long connection parameter names are used by default.

**See also**

- [“SAConnectionStringBuilderBase.GetUseLongNameAsKeyword method \[SQL Anywhere .NET\]” on page 201](#)

## ShouldSerialize method

Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

**Visual Basic syntax**

```
Public Overrides Function ShouldSerialize(  
    ByVal keyword As String  
) As Boolean
```

**C# syntax**

```
public override bool ShouldSerialize(string keyword)
```

**Parameters**

- **keyword** The key to locate in the SAConnectionStringBuilder.

**Returns**

True if the SAConnectionStringBuilder contains an entry with the specified key; otherwise false.

## TryGetValue method

Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

**Visual Basic syntax**

```
Public Overrides Function TryGetValue(  
    ByVal keyword As String,  
    ByVal value As Object  
) As Boolean
```

**C# syntax**

```
public override bool TryGetValue(string keyword, out object value)
```

### Parameters

- **keyword** The key of the item to retrieve.
- **value** The value corresponding to keyword.

### Returns

true if keyword was found within the connection string; otherwise false.

## Keys property

Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Keys As ICollection
```

### C# syntax

```
public override ICollection Keys {get;}
```

### Returns

An System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.

## this property

Gets or sets the value of the connection keyword.

### Visual Basic syntax

```
Public Overrides Property Item(ByVal keyword As String) As Object
```

### C# syntax

```
public override object this[string keyword] {get;set;}
```

### Parameters

- **keyword** The name of the connection keyword.

### Remarks

An object representing the value of the specified connection keyword.

If the keyword or type is invalid, an exception is raised. keyword is case insensitive.

When setting the value, passing NULL clears the value.

## SADDataAdapter class

Represents a set of commands and a database connection used to fill a `System.Data.DataSet` and to update a database.

### Visual Basic syntax

```
Public NotInheritable Class SADDataAdapter
    Inherits System.Data.Common.DbDataAdapter
    Implements System.ICloneable
```

### C# syntax

```
public sealed class SADDataAdapter :
    System.Data.Common.DbDataAdapter,
    System.ICloneable
```

### Base classes

- [System.Data.Common.DbDataAdapter](#)
- [System.ICloneable](#)

### Members

All members of `SADDataAdapter` class, including all inherited members.

Name	Description
<a href="#">SADDataAdapter constructor</a>	Initializes an <code>SADDataAdapter</code> object.
<a href="#">AddToBatch method</a> (Inherited from <code>System.Data.Common.DbDataAdapter</code> )	Adds a <a href="#">System.Data.IDbCommand</a> to the current batch.
<a href="#">CreateRowUpdatedEvent method</a> (Inherited from <code>System.Data.Common.DbDataAdapter</code> )	Initializes a new instance of the <a href="#">System.Data.Common.RowUpdatedEventArgs</a> class.
<a href="#">CreateRowUpdatingEvent method</a> (Inherited from <code>System.Data.Common.DbDataAdapter</code> )	Initializes a new instance of the <a href="#">System.Data.Common.RowUpdatingEventArgs</a> class.
<a href="#">Dispose method</a> (Inherited from <code>System.Data.Common.DbDataAdapter</code> )	Releases the unmanaged resources used by the <a href="#">System.Data.Common.DbDataAdapter</a> and optionally releases the managed resources.
<a href="#">ExecuteBatch method</a> (Inherited from <code>System.Data.Common.DbDataAdapter</code> )	Executes the current batch.

Name	Description
<a href="#">Fill method</a> (Inherited from System.Data.Common.DbDataAdapter)	Adds or refreshes rows in the <a href="#">System.Data.DataSet</a> .
<a href="#">FillSchema method</a> (Inherited from System.Data.Common.DbDataAdapter)	Adds a <a href="#">System.Data.DataTable</a> named "Table" to the specified <a href="#">System.Data.DataSet</a> and configures the schema to match that in the data source based on the specified <a href="#">System.Data.SchemaType</a> .
<a href="#">GetBatchedParameter method</a> (Inherited from System.Data.Common.DbDataAdapter)	Returns a <a href="#">System.Data.IDataParameter</a> from one of the commands in the current batch.
<a href="#">GetBatchedRecordsAffected method</a> (Inherited from System.Data.Common.DbDataAdapter)	Returns information about an individual update attempt within a larger batched update.
<a href="#">GetFillParameters method</a>	Returns the parameters set by you when executing a SELECT statement.
<a href="#">FillError</a> (Inherited from System.Data.Common.DbDataAdapter)  <a href="#">OnFillError method</a> (Inherited from System.Data.Common.DbDataAdapter)	
<a href="#">OnRowUpdated method</a> (Inherited from System.Data.Common.DbDataAdapter)	Raises the RowUpdated event of a .NET Framework data provider.
<a href="#">OnRowUpdating method</a> (Inherited from System.Data.Common.DbDataAdapter)	Raises the RowUpdating event of a .NET Framework data provider.
<a href="#">Update method</a> (Inherited from System.Data.Common.DbDataAdapter)	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of <a href="#">System.Data.DataRow</a> objects.
<a href="#">DeleteCommand property</a>	Specifies an SACommand object that is executed against the database when the Update method is called to delete rows in the database that correspond to deleted rows in the DataSet.
<a href="#">FillCommandBehavior property</a> (Inherited from System.Data.Common.DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.



Name	Description
<a href="#">InsertCommand property</a>	Specifies an SACommand that is executed against the database when the Update method is called that adds rows to the database to correspond to rows that were inserted in the DataSet.
<a href="#">SelectCommand property</a>	Specifies an SACommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.
<a href="#">TableMappings property</a>	Specifies a collection that provides the master mapping between a source table and a DataTable.
<a href="#">UpdateBatchSize property</a>	Gets or sets the number of rows that are processed in each round-trip to the server.
<a href="#">UpdateCommand property</a>	Specifies an SACommand that is executed against the database when the Update method is called to update rows in the database that correspond to updated rows in the DataSet.
<a href="#">RowUpdated event</a>	Occurs during an update after a command is executed against the data source.
<a href="#">RowUpdating event</a>	Occurs during an update before a command is executed against the data source.

### Remarks

The System.Data.DataSet provides a way to work with data offline. The SDataAdapter provides methods to associate a DataSet with a set of SQL statements.

**Implements:** IDbDataAdapter, IDataAdapter, ICloneable

For more information, see [“Using the SDataAdapter object to access and manipulate data”](#) on page 51 and [“Data access and manipulation”](#) on page 46.

## SDataAdapter constructor

Initializes an SDataAdapter object.

### Overload list

Name	Description
<a href="#">SDataAdapter() constructor</a>	Initializes an SDataAdapter object.

Name	Description
<a href="#">SADDataAdapter(SACommand) constructor</a>	Initializes an SADDataAdapter object with the specified SELECT statement.
<a href="#">SADDataAdapter(string, SAConnection) constructor</a>	Initializes an SADDataAdapter object with the specified SELECT statement and connection.
<a href="#">SADDataAdapter(string, string) constructor</a>	Initializes an SADDataAdapter object with the specified SELECT statement and connection string.

### SADDataAdapter() constructor

Initializes an SADDataAdapter object.

#### Visual Basic syntax

```
Public Sub New()
```

#### C# syntax

```
public SADDataAdapter()
```

#### See also

- [“SADDataAdapter.SADDataAdapter constructor \[SQL Anywhere .NET\]” on page 207](#)

### SADDataAdapter(SACommand) constructor

Initializes an SADDataAdapter object with the specified SELECT statement.

#### Visual Basic syntax

```
Public Sub New(ByVal selectCommand As SACommand)
```

#### C# syntax

```
public SADDataAdapter(SACommand selectCommand)
```

#### Parameters

- **selectCommand** An SACommand object that is used during `System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet)` to select records from the data source for placement in the `System.Data.DataSet`.

#### See also

- [“SADDataAdapter.SADDataAdapter constructor \[SQL Anywhere .NET\]” on page 207](#)

### SADDataAdapter(string, SAConnection) constructor

Initializes an SADDataAdapter object with the specified SELECT statement and connection.

**Visual Basic syntax**

```
Public Sub New(  
    ByVal selectCommandText As String,  
    ByVal selectConnection As SAConnection  
)
```

**C# syntax**

```
public SADDataAdapter(  
    string selectCommandText,  
    SAConnection selectConnection  
)
```

**Parameters**

- **selectCommandText** A SELECT statement to be used to set the SADDataAdapter.SelectCommand property of the SADDataAdapter object.
- **selectConnection** An SAConnection object that defines a connection to a database.

**See also**

- [“SADDataAdapter.SADDataAdapter constructor \[SQL Anywhere .NET\]” on page 207](#)
- [“SADDataAdapter.SelectCommand property \[SQL Anywhere .NET\]” on page 211](#)
- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)

**SADDataAdapter(string, string) constructor**

Initializes an SADDataAdapter object with the specified SELECT statement and connection string.

**Visual Basic syntax**

```
Public Sub New(  
    ByVal selectCommandText As String,  
    ByVal selectConnectionString As String  
)
```

**C# syntax**

```
public SADDataAdapter(  
    string selectCommandText,  
    string selectConnectionString  
)
```

**Parameters**

- **selectCommandText** A SELECT statement to be used to set the SADDataAdapter.SelectCommand property of the SADDataAdapter object.
- **selectConnectionString** A connection string for a SQL Anywhere database.

**See also**

- [“SADDataAdapter.SADDataAdapter constructor \[SQL Anywhere .NET\]” on page 207](#)
- [“SADDataAdapter.SelectCommand property \[SQL Anywhere .NET\]” on page 211](#)

## GetFillParameters method

Returns the parameters set by you when executing a SELECT statement.

### Visual Basic syntax

```
Public Shadows Function GetFillParameters() As SAPparameter()
```

### C# syntax

```
public new SAPparameter[] GetFillParameters()
```

### Returns

An array of IDataParameter objects that contains the parameters set by the user.

## DeleteCommand property

Specifies an SACommand object that is executed against the database when the Update method is called to delete rows in the database that correspond to deleted rows in the DataSet.

### Visual Basic syntax

```
Public Shadows Property DeleteCommand As SACommand
```

### C# syntax

```
public new SACommand DeleteCommand {get;set;}
```

### Remarks

If this property is not set and primary key information is present in the DataSet during Update, DeleteCommand can be generated automatically by setting SelectCommand and using the SACommandBuilder. In that case, the SACommandBuilder generates any additional commands that you do not set. This generation logic requires key column information to be present in the SelectCommand.

When DeleteCommand is assigned to an existing SACommand object, the SACommand object is not cloned. The DeleteCommand maintains a reference to the existing SACommand.

### See also

- [“SADDataAdapter.SelectCommand property \[SQL Anywhere .NET\]” on page 211](#)

## InsertCommand property

Specifies an SACommand that is executed against the database when the Update method is called that adds rows to the database to correspond to rows that were inserted in the DataSet.

### Visual Basic syntax

```
Public Shadows Property InsertCommand As SACommand
```

**C# syntax**

```
public new SACommand InsertCommand {get;set;}
```

**Remarks**

The SACommandBuilder does not require key columns to generate InsertCommand.

When InsertCommand is assigned to an existing SACommand object, the SACommand is not cloned. The InsertCommand maintains a reference to the existing SACommand.

If this command returns rows, the rows may be added to the DataSet depending on how you set the UpdatedRowSource property of the SACommand object.

## SelectCommand property

Specifies an SACommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.

**Visual Basic syntax**

```
Public Shadows Property SelectCommand As SACommand
```

**C# syntax**

```
public new SACommand SelectCommand {get;set;}
```

**Remarks**

When SelectCommand is assigned to a previously-created SACommand, the SACommand is not cloned. The SelectCommand maintains a reference to the previously-created SACommand object.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

The SELECT statement can also be specified in the SADataAdapter constructor.

## TableMappings property

Specifies a collection that provides the master mapping between a source table and a DataTable.

**Visual Basic syntax**

```
Public ReadOnly Shadows Property TableMappings As  
DataTableMappingCollection
```

**C# syntax**

```
public new DataTableMappingCollection TableMappings {get;}
```

**Remarks**

The default value is an empty collection.

When reconciling changes, the `SADDataAdapter` uses the `DataTableMappingCollection` collection to associate the column names used by the data source with the column names used by the `DataSet`.

The `TableMappings` property is not available in the .NET Compact Framework 2.0.

## UpdateBatchSize property

Gets or sets the number of rows that are processed in each round-trip to the server.

### Visual Basic syntax

```
Public Overrides Property UpdateBatchSize As Integer
```

### C# syntax

```
public override int UpdateBatchSize {get;set;}
```

### Remarks

The default value is 1.

Setting the value to something greater than 1 causes `SADDataAdapter.Update` to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of `UpdateBatchSize`. Setting the value to 0 causes `Update` to send the insert statements in a single batch.

Setting the value to something greater than 1 causes `SADDataAdapter.Fill` to execute all the insert statements in batches. The deletions and updates are executed sequentially as before, but insertions are executed afterward in batches of size equal to the value of `UpdateBatchSize`.

Setting the value to 0 causes `Fill` to send the insert statements in a single batch.

Setting it less than 0 is an error.

If `UpdateBatchSize` is set to something other than one, and the `InsertCommand` property is set to something that is not an `INSERT` statement, then an exception is thrown when calling `Fill`.

This behavior is different from `SqlDataAdapter`. It batches all types of commands.

## UpdateCommand property

Specifies an `SACCommand` that is executed against the database when the `Update` method is called to update rows in the database that correspond to updated rows in the `DataSet`.

### Visual Basic syntax

```
Public Shadows Property UpdateCommand As SACCommand
```

### C# syntax

```
public new SACCommand UpdateCommand {get;set;}
```

**Remarks**

During Update, if this property is not set and primary key information is present in the SelectCommand, the UpdateCommand can be generated automatically if you set the SelectCommand property and use the SACommandBuilder. Then, any additional commands that you do not set are generated by the SACommandBuilder. This generation logic requires key column information to be present in the SelectCommand.

When UpdateCommand is assigned to a previously-created SACommand, the SACommand is not cloned. The UpdateCommand maintains a reference to the previously-created SACommand object.

If execution of this command returns rows, these rows can be merged with the DataSet depending on how you set the UpdatedRowSource property of the SACommand object.

## RowUpdated event

Occurs during an update after a command is executed against the data source.

**Visual Basic syntax**

```
Public Event RowUpdated As SARowUpdatedEventHandler
```

**C# syntax**

```
public event SARowUpdatedEventHandler RowUpdated;
```

**Remarks**

When an attempt to update is made, the event fires.

The event handler receives an argument of type SARowUpdatedEventArgs containing data related to this event.

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdated Event.

## RowUpdating event

Occurs during an update before a command is executed against the data source.

**Visual Basic syntax**

```
Public Event RowUpdating As SARowUpdatingEventHandler
```

**C# syntax**

```
public event SARowUpdatingEventHandler RowUpdating;
```

**Remarks**

When an attempt to update is made, the event fires.

The event handler receives an argument of type `SARowUpdatingEventArgs` containing data related to this event.

For more information, see the .NET Framework documentation for `OleDbDataAdapter.RowUpdating` Event.

## SADataReader class

A read-only, forward-only result set from a query or stored procedure.

### Visual Basic syntax

```
Public NotInheritable Class SADataReader
    Inherits System.Data.Common.DbDataReader
    Implements System.ComponentModel.IListSource
```

### C# syntax

```
public sealed class SADataReader :
    System.Data.Common.DbDataReader,
    System.ComponentModel.IListSource
```

### Base classes

- [System.Data.Common.DbDataReader](#)
- [System.ComponentModel.IListSource](#)

### Members

All members of `SADataReader` class, including all inherited members.

Name	Description
<a href="#">Close method</a>	Closes the <code>SADataReader</code> .
<a href="#">Dispose method</a> (Inherited from <code>System.Data.Common.DbDataReader</code> )	Releases all resources used by the current instance of the <a href="#">System.Data.Common.DbDataReader</a> class.
<a href="#">GetBoolean method</a>	Returns the value of the specified column as a Boolean.
<a href="#">GetByte method</a>	Returns the value of the specified column as a Byte.
<a href="#">GetBytes method</a>	Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.
<a href="#">GetChar method</a>	Returns the value of the specified column as a character.
<a href="#">GetChars method</a>	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.



Name	Description
<a href="#">GetData method</a>	This method is not supported.
<a href="#">GetDataTypeName method</a>	Returns the name of the source data type.
<a href="#">GetDateTime method</a>	Returns the value of the specified column as a DateTime object.
<a href="#">GetDateTimeOffset method</a>	Returns the value of the specified column as a DateTimeOffset object.
<a href="#">GetDbDataReader method</a> (Inherited from System.Data.Common.DbDataReader)	Returns a <a href="#">System.Data.Common.DbDataReader</a> object for the requested column ordinal that can be overridden with a provider-specific implementation.
<a href="#">GetDecimal method</a>	Returns the value of the specified column as a Decimal object.
<a href="#">GetDouble method</a>	Returns the value of the specified column as a double-precision floating-point number.
<a href="#">GetEnumerator method</a>	Returns a System.Collections.IEnumerator that iterates through the SADataReader object.
<a href="#">GetFieldType method</a>	Returns the Type that is the data type of the object.
<a href="#">GetFloat method</a>	Returns the value of the specified column as a single-precision floating-point number.
<a href="#">GetGuid method</a>	Returns the value of the specified column as a global unique identifier (GUID).
<a href="#">GetInt16 method</a>	Returns the value of the specified column as a 16-bit signed integer.
<a href="#">GetInt32 method</a>	Returns the value of the specified column as a 32-bit signed integer.
<a href="#">GetInt64 method</a>	Returns the value of the specified column as a 64-bit signed integer.
<a href="#">GetName method</a>	Returns the name of the specified column.
<a href="#">GetOrdinal method</a>	Returns the column ordinal, given the column name.
<a href="#">GetProviderSpecificFieldType method</a> (Inherited from System.Data.Common.DbDataReader)	Returns the provider-specific field type of the specified column.

Name	Description
<a href="#">GetProviderSpecificValue method</a> (Inherited from System.Data.Common.DbDataReader)	Gets the value of the specified column as an instance of <a href="#">System.Object</a> .
<a href="#">GetProviderSpecificValues method</a> (Inherited from System.Data.Common.DbDataReader)	Gets all provider-specific attribute columns in the collection for the current row.
<a href="#">GetSchemaTable method</a>	Returns a DataTable that describes the column metadata of the SADataReader.
<a href="#">GetString method</a>	Returns the value of the specified column as a string.
<a href="#">GetTimeSpan method</a>	Returns the value of the specified column as a TimeSpan object.
<a href="#">GetUInt16 method</a>	Returns the value of the specified column as a 16-bit unsigned integer.
<a href="#">GetUInt32 method</a>	Returns the value of the specified column as a 32-bit unsigned integer.
<a href="#">GetUInt64 method</a>	Returns the value of the specified column as a 64-bit unsigned integer.
<a href="#">GetValue method</a>	Returns the value of the specified column as an Object.
<a href="#">GetValues method</a>	Gets all the columns in the current row.
<a href="#">IsDBNull method</a>	Returns a value indicating whether the column contains NULL values.
<a href="#">myDispose method</a>	Frees the resources associated with the object.
<a href="#">NextResult method</a>	Advances the SADataReader to the next result, when reading the results of batch SQL statements.
<a href="#">Read method</a>	Reads the next row of the result set and moves the SADataReader to that row.
<a href="#">Depth property</a>	Gets a value indicating the depth of nesting for the current row.
<a href="#">FieldCount property</a>	Gets the number of columns in the result set.
<a href="#">HasRows property</a>	Gets a value that indicates whether the SADataReader contains one or more rows.

Name	Description
<a href="#">IsClosed property</a>	Gets a values that indicates whether the SADataReader is closed.
<a href="#">RecordsAffected property</a>	The number of rows changed, inserted, or deleted by execution of the SQL statement.
<a href="#">this property</a>	Returns the value of a column in its native format.
<a href="#">VisibleFieldCount property</a> (Inherited from System.Data.Common.DbDataReader)	Gets the number of fields in the <a href="#">System.Data.Common.DbDataReader</a> that are not hidden.

### Remarks

There is no constructor for SADataReader. To get an SADataReader object, execute an SACommand:

```
SACommand cmd = new SACommand(
    "SELECT EmployeeID FROM Employees", conn );
SADataReader reader = cmd.ExecuteReader();
```

You can only move forward through an SADataReader. If you need a more flexible object to manipulate results, use an SADataAdapter.

The SADataReader retrieves rows as needed, whereas the SADataAdapter must retrieve all rows of a result set before you can carry out any action on the object. For large result sets, this difference gives the SADataReader a much faster response time.

**Implements:** IDataReader, IDisposable, IDataRecord, IListSource

For more information, see [“Data access and manipulation”](#) on page 46.

### See also

- [“SACommand.ExecuteReader method \[SQL Anywhere .NET\]”](#) on page 137

## Close method

Closes the SADataReader.

### Visual Basic syntax

```
Public Overrides Sub Close()
```

### C# syntax

```
public override void Close()
```

### Remarks

You must explicitly call the Close method when you are finished using the SADataReader.

When running in autocommit mode, a COMMIT is issued as a side effect of closing the SADataReader.

## GetBoolean method

Returns the value of the specified column as a Boolean.

### Visual Basic syntax

```
Public Overrides Function GetBoolean(  
    ByVal ordinal As Integer  
) As Boolean
```

### C# syntax

```
public override bool GetBoolean(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the column.

### Remarks

No conversions are performed, so the data retrieved must already be a Boolean.

### See also

- [“SADaReader.GetOrdinal method \[SQL Anywhere .NET\]” on page 228](#)
- [“SADaReader.GetFieldType method \[SQL Anywhere .NET\]” on page 225](#)

## GetByte method

Returns the value of the specified column as a Byte.

### Visual Basic syntax

```
Public Overrides Function GetByte(ByVal ordinal As Integer) As Byte
```

### C# syntax

```
public override byte GetByte(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the column.

## Remarks

No conversions are performed, so the data retrieved must already be a byte.

## GetBytes method

Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.

### Visual Basic syntax

```
Public Overrides Function GetBytes(  
    ByVal ordinal As Integer,  
    ByVal dataIndex As Long,  
    ByVal buffer As Byte(),  
    ByVal bufferIndex As Integer,  
    ByVal length As Integer  
    ) As Long
```

### C# syntax

```
public override long GetBytes(  
    int ordinal,  
    long dataIndex,  
    byte[] buffer,  
    int bufferIndex,  
    int length  
    )
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.
- **dataIndex** The index within the column value from which to read bytes.
- **buffer** An array in which to store the data.
- **bufferIndex** The index in the array to start copying data.
- **length** The maximum length to copy into the specified buffer.

### Returns

The number of bytes read.

### Remarks

GetBytes returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This may be the case, for example, when the SADataReader is reading a large data structure into a buffer.

If you pass a buffer that is a null reference (Nothing in Visual Basic), GetBytes returns the length of the field in bytes.

No conversions are performed, so the data retrieved must already be a byte array.

## GetChar method

Returns the value of the specified column as a character.

### Visual Basic syntax

```
Public Overrides Function GetChar(ByVal ordinal As Integer) As Char
```

### C# syntax

```
public override char GetChar(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the column.

### Remarks

No conversions are performed, so the data retrieved must already be a character.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetChars method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

### Visual Basic syntax

```
Public Overrides Function GetChars(  
    ByVal ordinal As Integer,  
    ByVal dataIndex As Long,  
    ByVal buffer As Char(),  
    ByVal bufferIndex As Integer,  
    ByVal length As Integer  
) As Long
```

### C# syntax

```
public override long GetChars(  
    int ordinal,  
    long dataIndex,
```

```
        char[] buffer,  
        int bufferIndex,  
        int length  
    )
```

### Parameters

- **ordinal** The zero-based column ordinal.
- **dataIndex** The index within the row from which to begin the read operation.
- **buffer** The buffer into which to copy data.
- **bufferIndex** The index for buffer to begin the read operation.
- **length** The number of characters to read.

### Returns

The actual number of characters read.

### Remarks

GetChars returns the number of available characters in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetChars has already been used to obtain characters from the field. This may be the case, for example, when the SADataReader is reading a large data structure into a buffer.

If you pass a buffer that is a null reference (Nothing in Visual Basic), GetChars returns the length of the field in characters.

No conversions are performed, so the data retrieved must already be a character array.

For information about handling BLOBs, see [“BLOBs” on page 60](#).

## GetData method

This method is not supported.

### Visual Basic syntax

```
Public Shadows Function GetData(ByVal i As Integer) As IDataReader
```

### C# syntax

```
public new IDataReader GetData(int i)
```

### Remarks

When called, it throws an InvalidOperationException.

### See also

- [System.InvalidOperationException](#)

## GetDataTypeName method

Returns the name of the source data type.

### Visual Basic syntax

```
Public Overrides Function GetDataTypeName(  
    ByVal index As Integer  
) As String
```

### C# syntax

```
public override string GetDataTypeName(int index)
```

### Parameters

- **index** The zero-based column ordinal.

### Returns

The name of the back-end data type.

## GetDateTime method

Returns the value of the specified column as a DateTime object.

### Visual Basic syntax

```
Public Overrides Function GetDateTime(ByVal ordinal As Integer) As Date
```

### C# syntax

```
public override DateTime GetDateTime(int ordinal)
```

### Parameters

- **ordinal** The zero-based column ordinal.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a DateTime object.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)



## GetDateTimeOffset method

Returns the value of the specified column as a DateTimeOffset object.

### Visual Basic syntax

```
Public Function GetDateTimeOffset(  
    ByVal ordinal As Integer  
) As DateTimeOffset
```

### C# syntax

```
public DateTimeOffset GetDateTimeOffset(int ordinal)
```

### Parameters

- **ordinal** The zero-based column ordinal.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a DateTimeOffset object.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetDecimal method

Returns the value of the specified column as a Decimal object.

### Visual Basic syntax

```
Public Overrides Function GetDecimal(  
    ByVal ordinal As Integer  
) As Decimal
```

### C# syntax

```
public override decimal GetDecimal(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a Decimal object.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetDouble method

Returns the value of the specified column as a double-precision floating-point number.

### Visual Basic syntax

```
Public Overrides Function GetDouble(ByVal ordinal As Integer) As Double
```

### C# syntax

```
public override double GetDouble(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a double-precision floating-point number.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetEnumerator method

Returns a `System.Collections.IEnumerator` that iterates through the `SADataReader` object.

### Visual Basic syntax

```
Public Overrides Function GetEnumerator()  
    As System.Collections.IEnumerator
```

### C# syntax

```
public override IEnumerator GetEnumerator()
```

**Returns**

A System.Collections.IEnumerator for the SADATAReader object.

**See also**

- [“SADATAReader class \[SQL Anywhere .NET\]” on page 214](#)

## GetFieldType method

Returns the Type that is the data type of the object.

**Visual Basic syntax**

```
Public Overrides Function GetFieldType(ByVal index As Integer) As Type
```

**C# syntax**

```
public override Type GetFieldType(int index)
```

**Parameters**

- **index** The zero-based column ordinal.

**Returns**

The type that is the data type of the object.

## GetFloat method

Returns the value of the specified column as a single-precision floating-point number.

**Visual Basic syntax**

```
Public Overrides Function GetFloat(ByVal ordinal As Integer) As Single
```

**C# syntax**

```
public override float GetFloat(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

No conversions are performed, so the data retrieved must already be a single-precision floating-point number.

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

#### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetGuid method

Returns the value of the specified column as a global unique identifier (GUID).

#### Visual Basic syntax

```
Public Overrides Function GetGuid(ByVal ordinal As Integer) As Guid
```

#### C# syntax

```
public override Guid GetGuid(int ordinal)
```

#### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

#### Returns

The value of the specified column.

#### Remarks

The data retrieved must already be a globally-unique identifier or binary(16).

Call the `SADataReader.IsDBNull` method to check for null values before calling this method.

#### See also

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetInt16 method

Returns the value of the specified column as a 16-bit signed integer.

#### Visual Basic syntax

```
Public Overrides Function GetInt16(ByVal ordinal As Integer) As Short
```

#### C# syntax

```
public override short GetInt16(int ordinal)
```

#### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

No conversions are performed, so the data retrieved must already be a 16-bit signed integer.

## GetInt32 method

Returns the value of the specified column as a 32-bit signed integer.

**Visual Basic syntax**

```
Public Overrides Function GetInt32(ByVal ordinal As Integer) As Integer
```

**C# syntax**

```
public override int GetInt32(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

No conversions are performed, so the data retrieved must already be a 32-bit signed integer.

## GetInt64 method

Returns the value of the specified column as a 64-bit signed integer.

**Visual Basic syntax**

```
Public Overrides Function GetInt64(ByVal ordinal As Integer) As Long
```

**C# syntax**

```
public override long GetInt64(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

## Remarks

No conversions are performed, so the data retrieved must already be a 64-bit signed integer.

## GetName method

Returns the name of the specified column.

### Visual Basic syntax

```
Public Overrides Function GetName(ByVal index As Integer) As String
```

### C# syntax

```
public override string GetName(int index)
```

### Parameters

- **index** The zero-based index of the column.

### Returns

The name of the specified column.

## GetOrdinal method

Returns the column ordinal, given the column name.

### Visual Basic syntax

```
Public Overrides Function GetOrdinal(ByVal name As String) As Integer
```

### C# syntax

```
public override int GetOrdinal(string name)
```

### Parameters

- **name** The column name.

### Returns

The zero-based column ordinal.

### Remarks

GetOrdinal performs a case-sensitive lookup first. If it fails, a second case-insensitive search is made.

GetOrdinal is Japanese kana-width insensitive.

Because ordinal-based lookups are more efficient than named lookups, it is inefficient to call GetOrdinal within a loop. You can save time by calling GetOrdinal once and assigning the results to an integer variable for use within the loop.

## GetSchemaTable method

Returns a DataTable that describes the column metadata of the SADataReader.

### Visual Basic syntax

```
Public Overrides Function GetSchemaTable() As DataTable
```

### C# syntax

```
public override DataTable GetSchemaTable()
```

### Returns

A DataTable that describes the column metadata.

### Remarks

This method returns metadata about each column in the following order:

DataTable column	Description
ColumnName	The name of the column or a null reference (Nothing in Visual Basic) if the column has no name. If the column is aliased in the SQL query, the alias is returned. Note that in result sets, not all columns have names and not all column names are unique.
ColumnOrdinal	The ID of the column. The value is in the range [0, FieldCount -1].
ColumnSize	For sized columns, the maximum length of a value in the column. For other columns, this is the size in bytes of the data type.
NumericPrecision	The precision of a numeric column or DBNull if the column is not numeric.
NumericScale	The scale of a numeric column or DBNull if the column is not numeric.
IsUnique	True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.
IsKey	True if the column is one of a set of columns in the result set that taken together from a unique key for the result set. The set of columns with IsKey set to true does not need to be the minimal set that uniquely identifies a row in the result set.
BaseServerName	The name of the SQL Anywhere database server used by the SADataReader.
BaseCatalogName	The name of the catalog in the database that contains the column. This value is always DBNull.
BaseColumnName	The original name of the column in the table BaseTableName of the database or DBNull if the column is computed or if this information cannot be determined.

<b>DataTable column</b>	<b>Description</b>
BaseSchemaName	The name of the schema in the database that contains the column.
BaseTableName	The name of the table in the database that contains the column, or DBNull if column is computed or if this information cannot be determined.
DataType	The .NET data type that is most appropriate for this type of column.
AllowDBNull	True if the column is nullable, false if the column is not nullable or if this information cannot be determined.
ProviderType	The type of the column.
IsAliased	True if the column name is an alias, false if it is not an alias.
IsExpression	True if the column is an expression, false if it is a column value.
IsIdentity	True if the column is an identity column, false if it is not an identity column.
IsAutoIncrement	True if the column is an autoincrement or global autoincrement column, false otherwise (or if this information cannot be determined).
IsRowVersion	True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identify the row.
IsHidden	True if the column is hidden, false otherwise.
IsLong	True if the column is a long varchar, long nvarchar, or a long binary column, false otherwise.
IsReadOnly	True if the column is read-only, false if the column is modifiable or if its access cannot be determined.

For more information about these columns, see the .NET Framework documentation for `SqlDataReader.GetSchemaTable`.

For more information, see [“Result set schema information” on page 50](#).

## GetString method

Returns the value of the specified column as a string.

### Visual Basic syntax

```
Public Overrides Function GetString(ByVal ordinal As Integer) As String
```



**C# syntax**

```
public override string GetString(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

No conversions are performed, so the data retrieved must already be a string.

Call the `SADataReader.IsDBNull` method to check for NULL values before calling this method.

**See also**

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetTimeSpan method

Returns the value of the specified column as a `TimeSpan` object.

**Visual Basic syntax**

```
Public Function GetTimeSpan(ByVal ordinal As Integer) As TimeSpan
```

**C# syntax**

```
public TimeSpan GetTimeSpan(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

The column must be a SQL Anywhere TIME data type. The data is converted to `TimeSpan`. The `Days` property of `TimeSpan` is always set to 0.

Call `SADataReader.IsDBNull` method to check for NULL values before calling this method.

For more information, see [“Time values” on page 60](#).

**See also**

- [“SADataReader.IsDBNull method \[SQL Anywhere .NET\]” on page 235](#)

## GetUInt16 method

Returns the value of the specified column as a 16-bit unsigned integer.

### Visual Basic syntax

```
Public Function GetUInt16(ByVal ordinal As Integer) As UShort
```

### C# syntax

```
public ushort GetUInt16(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a 16-bit unsigned integer.

## GetUInt32 method

Returns the value of the specified column as a 32-bit unsigned integer.

### Visual Basic syntax

```
Public Function GetUInt32(ByVal ordinal As Integer) As UInteger
```

### C# syntax

```
public uint GetUInt32(int ordinal)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

### Returns

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a 32-bit unsigned integer.

## GetUInt64 method

Returns the value of the specified column as a 64-bit unsigned integer.

**Visual Basic syntax**

```
Public Function GetUInt64(ByVal ordinal As Integer) As ULong
```

**C# syntax**

```
public ulong GetUInt64(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column.

**Remarks**

No conversions are performed, so the data retrieved must already be a 64-bit unsigned integer.

## GetValue method

Returns the value of the specified column as an Object.

**Overload list**

Name	Description
<a href="#">GetValue(int) method</a>	Returns the value of the specified column as an Object.
<a href="#">GetValue(int, long, int) method</a>	Returns a substring of the value of the specified column as an Object.

### GetValue(int) method

Returns the value of the specified column as an Object.

**Visual Basic syntax**

```
Public Overrides Function GetValue(ByVal ordinal As Integer) As Object
```

**C# syntax**

```
public override object GetValue(int ordinal)
```

**Parameters**

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

**Returns**

The value of the specified column as an object.

## Remarks

This method returns DBNull for NULL database columns.

## GetValue(int, long, int) method

Returns a substring of the value of the specified column as an Object.

### Visual Basic syntax

```
Public Function GetValue(  
    ByVal ordinal As Integer,  
    ByVal index As Long,  
    ByVal length As Integer  
) As Object
```

### C# syntax

```
public object GetValue(int ordinal, long index, int length)
```

### Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.
- **index** A zero-based index of the substring of the value to be obtained.
- **length** The length of the substring of the value to be obtained.

### Returns

The substring value is returned as an object.

### Remarks

This method returns DBNull for NULL database columns.

## GetValues method

Gets all the columns in the current row.

### Visual Basic syntax

```
Public Overrides Function GetValues(ByVal values As Object()) As Integer
```

### C# syntax

```
public override int GetValues(object[] values)
```

### Parameters

- **values** An array of objects that holds an entire row of the result set.

**Returns**

The number of objects in the array.

**Remarks**

For most applications, the `GetValues` method provides an efficient means for retrieving all columns, rather than retrieving each column individually.

You can pass an Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an Object array whose length is more than the number of columns contained in the resulting row.

This method returns `DBNull` for `NULL` database columns.

## IsDBNull method

Returns a value indicating whether the column contains `NULL` values.

**Visual Basic syntax**

```
Public Overrides Function IsDBNull(ByVal ordinal As Integer) As Boolean
```

**C# syntax**

```
public override bool IsDBNull(int ordinal)
```

**Parameters**

- **ordinal** The zero-based column ordinal.

**Returns**

Returns true if the specified column value is equivalent to `DBNull`. Otherwise, it returns false.

**Remarks**

Call this method to check for `NULL` column values before calling the typed get methods (for example, `GetByte`, `GetChar`, and so on) to avoid raising an exception.

## myDispose method

Frees the resources associated with the object.

**Visual Basic syntax**

```
Public Sub myDispose()
```

**C# syntax**

```
public void myDispose()
```

## NextResult method

Advances the `SADataReader` to the next result, when reading the results of batch SQL statements.

### Visual Basic syntax

```
Public Overrides Function NextResult() As Boolean
```

### C# syntax

```
public override bool NextResult()
```

### Returns

Returns true if there are more result sets. Otherwise, it returns false.

### Remarks

Used to process multiple results, which can be generated by executing batch SQL statements.

By default, the data reader is positioned on the first result.

## Read method

Reads the next row of the result set and moves the `SADataReader` to that row.

### Visual Basic syntax

```
Public Overrides Function Read() As Boolean
```

### C# syntax

```
public override bool Read()
```

### Returns

Returns true if there are more rows. Otherwise, it returns false.

### Remarks

The default position of the `SADataReader` is prior to the first record. Therefore, you must call `Read` to begin accessing any data.

### Example

The following code fills a listbox with the values in a single column of results.

```
while( reader.Read() ) {  
    listResults.Items.Add( reader.GetValue( 0 ).ToString() );  
}  
listResults.EndUpdate();  
reader.Close();
```

## Depth property

Gets a value indicating the depth of nesting for the current row.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Depth As Integer
```

### C# syntax

```
public override int Depth {get;}
```

### Remarks

The outermost table has a depth of zero.

The depth of nesting for the current row.

## FieldCount property

Gets the number of columns in the result set.

### Visual Basic syntax

```
Public ReadOnly Overrides Property FieldCount As Integer
```

### C# syntax

```
public override int FieldCount {get;}
```

### Remarks

The number of columns in the current record.

## HasRows property

Gets a value that indicates whether the SADATAReader contains one or more rows.

### Visual Basic syntax

```
Public ReadOnly Overrides Property HasRows As Boolean
```

### C# syntax

```
public override bool HasRows {get;}
```

### Remarks

True if the SADATAReader contains one or more rows; otherwise, false.

## IsClosed property

Gets a values that indicates whether the SDataReader is closed.

### Visual Basic syntax

```
Public ReadOnly Overrides Property IsClosed As Boolean
```

### C# syntax

```
public override bool IsClosed {get;}
```

### Remarks

True if the SDataReader is closed; otherwise, false.

IsClosed and RecordsAffected are the only properties that you can call after the SDataReader is closed.

## RecordsAffected property

The number of rows changed, inserted, or deleted by execution of the SQL statement.

### Visual Basic syntax

```
Public ReadOnly Overrides Property RecordsAffected As Integer
```

### C# syntax

```
public override int RecordsAffected {get;}
```

### Remarks

The number of rows changed, inserted, or deleted. This is 0 if no rows were affected or the statement failed, or -1 for SELECT statements.

The number of rows changed, inserted, or deleted. The value is 0 if no rows were affected or the statement failed, and -1 for SELECT statements.

The value of this property is cumulative. For example, if two records are inserted in batch mode, the value of RecordsAffected will be two.

IsClosed and RecordsAffected are the only properties that you can call after the SDataReader is closed.

## this property

Returns the value of a column in its native format.



**Overload list**

Name	Description
<a href="#">this[int] property</a>	Returns the value of a column in its native format.
<a href="#">this[string] property</a>	Returns the value of a column in its native format.

**this[int] property**

Returns the value of a column in its native format.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property Item(  
    ByVal index As Integer  
) As Object
```

**C# syntax**

```
public override object this[int index] {get;}
```

**Parameters**

- **index** The column ordinal.

**Remarks**

In C#, this property is the indexer for the `SADatReader` class.

**this[string] property**

Returns the value of a column in its native format.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property Item(ByVal name As String) As Object
```

**C# syntax**

```
public override object this[string name] {get;}
```

**Parameters**

- **name** The column name.

**Remarks**

In C#, this property is the indexer for the `SADatReader` class.

## SADataSourceEnumerator class

Provides a mechanism for enumerating all available instances of SQL Anywhere database servers within the local network.

### Visual Basic syntax

```
Public NotInheritable Class SADataSourceEnumerator  
    Inherits System.Data.Common.DbDataSourceEnumerator
```

### C# syntax

```
public sealed class SADataSourceEnumerator :  
    System.Data.Common.DbDataSourceEnumerator
```

### Base classes

- [System.Data.Common.DbDataSourceEnumerator](#)

### Members

All members of SADataSourceEnumerator class, including all inherited members.

Name	Description
<a href="#">GetDataSources method</a>	Retrieves a DataTable containing information about all visible SQL Anywhere database servers.
<a href="#">Instance property</a>	Gets an instance of SADataSourceEnumerator, which can be used to retrieve information about all visible SQL Anywhere database servers.

### Remarks

There is no constructor for SADataSourceEnumerator.

The SADataSourceEnumerator class is not available in the .NET Compact Framework 2.0.

## GetDataSources method

Retrieves a DataTable containing information about all visible SQL Anywhere database servers.

### Visual Basic syntax

```
Public Overrides Function GetDataSources() As DataTable
```

### C# syntax

```
public override DataTable GetDataSources()
```

### Remarks

The returned table has four columns: ServerName, IPAddress, PortNumber, and DataBaseNames. There is a row in the table for each available database server.

## Example

The following code fills a DataTable with information for each database server that is available.

```
DataTable servers = SDataSourceEnumerator.Instance.GetDataSources();
```

## Instance property

Gets an instance of SDataSourceEnumerator, which can be used to retrieve information about all visible SQL Anywhere database servers.

### Visual Basic syntax

```
Public Shared ReadOnly Property Instance As SDataSourceEnumerator
```

### C# syntax

```
public SDataSourceEnumerator Instance {get;}
```

## SADefault class

Represents a parameter with a default value.

### Visual Basic syntax

```
Public NotInheritable Class SADefault
```

### C# syntax

```
public sealed class SADefault
```

### Members

All members of SADefault class, including all inherited members.

Name	Description
<a href="#">Value field</a>	Gets the value for a default parameter.

### Remarks

There is no constructor for SADefault.

```
SAParameter parm = new SAParameter();
parm.Value = SADefault.Value;
```

## Value field

Gets the value for a default parameter.

### Visual Basic syntax

```
Public Shared ReadOnly value As SADefault
```

### C# syntax

```
public static readonly SADefault value;
```

### Remarks

This field is read-only and static.

## SAError class

Collects information relevant to a warning or error returned by the data source.

### Visual Basic syntax

```
Public NotInheritable Class SAError
```

### C# syntax

```
public sealed class SAError
```

### Members

All members of SAError class, including all inherited members.

Name	Description
<a href="#">ToString method</a>	The complete text of the error message.
<a href="#">Message property</a>	Returns a short description of the error.
<a href="#">NativeError property</a>	Returns database-specific error information.
<a href="#">Source property</a>	Returns the name of the provider that generated the error.
<a href="#">SqlState property</a>	The SQL Anywhere five-character SQLSTATE following the ANSI SQL standard.

### Remarks

There is no constructor for SAError.

For information about error handling, see [“Error handling and the SQL Anywhere .NET Data Provider” on page 63](#).

## ToToString method

The complete text of the error message.

**Visual Basic syntax**

```
Public Overrides Function ToString() As String
```

**C# syntax**

```
public override string ToString()
```

**Example**

The return value is a string in the form **SAError:**, followed by the Message. For example:

```
SAError:UserId or Password not valid.
```

## Message property

Returns a short description of the error.

**Visual Basic syntax**

```
Public ReadOnly Property Message As String
```

**C# syntax**

```
public string Message {get;}
```

## NativeError property

Returns database-specific error information.

**Visual Basic syntax**

```
Public ReadOnly Property NativeError As Integer
```

**C# syntax**

```
public int NativeError {get;}
```

## Source property

Returns the name of the provider that generated the error.

**Visual Basic syntax**

```
Public ReadOnly Property Source As String
```

**C# syntax**

```
public string Source {get;}
```

## SqlState property

The SQL Anywhere five-character SQLSTATE following the ANSI SQL standard.

### Visual Basic syntax

```
Public ReadOnly Property sqlState As String
```

### C# syntax

```
public string sqlState {get;}
```

## SAErrorCollection class

Collects all errors generated by the SQL Anywhere .NET Data Provider.

### Visual Basic syntax

```
Public NotInheritable Class SAErrorCollection  
    Implements System.Collections.ICollection  
    Implements System.Collections.IEnumerable
```

### C# syntax

```
public sealed class SAErrorCollection :  
    System.Collections.ICollection,  
    System.Collections.IEnumerable
```

### Base classes

- [System.Collections.ICollection](#)
- [System.Collections.IEnumerable](#)

### Members

All members of SAErrorCollection class, including all inherited members.

Name	Description
<a href="#">CopyTo method</a>	Copies the elements of the SAErrorCollection into an array, starting at the given index within the array.
<a href="#">GetEnumerator method</a>	Returns an enumerator that iterates through the SAErrorCollection.
<a href="#">Count property</a>	Returns the number of errors in the collection.
<a href="#">this property</a>	Returns the error at the specified index.

### Remarks

There is no constructor for SAErrorCollection. Typically, an SAErrorCollection is obtained from the SAException.Errors property.

**Implements:** ICollection, IEnumerable

For information about error handling, see [“Error handling and the SQL Anywhere .NET Data Provider” on page 63](#).

**See also**

- [“SAException.Errors property \[SQL Anywhere .NET\]” on page 248](#)

## CopyTo method

Copies the elements of the SAErrorCollection into an array, starting at the given index within the array.

**Visual Basic syntax**

```
Public Sub CopyTo(ByVal array As Array, ByVal index As Integer)
```

**C# syntax**

```
public void CopyTo(Array array, int index)
```

**Parameters**

- **array** The array into which to copy the elements.
- **index** The starting index of the array.

## GetEnumerator method

Returns an enumerator that iterates through the SAErrorCollection.

**Visual Basic syntax**

```
Public Function GetEnumerator() As System.Collections.IEnumerator
```

**C# syntax**

```
public System.Collections.IEnumerator GetEnumerator()
```

**Returns**

An System.Collections.IEnumerator for the SAErrorCollection.

## Count property

Returns the number of errors in the collection.

**Visual Basic syntax**

```
Public ReadOnly Property Count As Integer
```

### C# syntax

```
public int Count {get;}
```

## this property

Returns the error at the specified index.

### Visual Basic syntax

```
Public ReadOnly Property Item(ByVal index As Integer) As SAError
```

### C# syntax

```
public SAError this[int index] {get;}
```

### Parameters

- **index** The zero-based index of the error to retrieve.

### Remarks

An SAError object that contains the error at the specified index.

### See also

- [“SAError class \[SQL Anywhere .NET\]” on page 242](#)

## SAException class

The exception that is thrown when SQL Anywhere returns a warning or error.

### Visual Basic syntax

```
Public Class SAException Inherits System.Exception
```

### C# syntax

```
public class SAException : System.Exception
```

### Base classes

- [System.Exception](#)

### Members

All members of SAException class, including all inherited members.



Name	Description
<a href="#">GetBaseException method</a> (Inherited from System.Exception)	When overridden in a derived class, returns the <a href="#">System.Exception</a> that is the root cause of one or more subsequent exceptions.
<a href="#">GetObjectData method</a>	Sets the <a href="#">SerializationInfo</a> with information about the exception.
<a href="#">GetType method</a> (Inherited from System.Exception)	Gets the runtime type of the current instance.
<a href="#">ToString method</a> (Inherited from System.Exception)	Creates and returns a string representation of the current exception.
<a href="#">Data property</a> (Inherited from System.Exception)	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
<a href="#">Errors property</a>	Returns a collection of one or more <a href="#">SAError</a> objects.
<a href="#">HelpLink property</a> (Inherited from System.Exception)	Gets or sets a link to the help file associated with this exception.
<a href="#">HRESULT property</a> (Inherited from System.Exception)	Gets or sets <a href="#">HRESULT</a> , a coded numerical value that is assigned to a specific exception.
<a href="#">InnerException property</a> (Inherited from System.Exception)	Gets the <a href="#">System.Exception</a> instance that caused the current exception.
<a href="#">Message property</a>	Returns the text describing the error.
<a href="#">NativeError property</a>	Returns database-specific error information.
<a href="#">Source property</a>	Returns the name of the provider that generated the error.
<a href="#">StackTrace property</a> (Inherited from System.Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
<a href="#">TargetSite property</a> (Inherited from System.Exception)	Gets the method that throws the current exception.

## Remarks

There is no constructor for [SAException](#). Typically, an [SAException](#) object is declared in a catch. For example:

```

...
catch( SAException ex )
{
    MessageBox.Show( ex.Errors[0].Message, "Error" );
}

```

For information about error handling, see [“Error handling and the SQL Anywhere .NET Data Provider” on page 63](#).

## GetObjectData method

Sets the `SerializationInfo` with information about the exception.

### Visual Basic syntax

```
Public Overrides Sub GetObjectData(  
    ByVal info As SerializationInfo,  
    ByVal context As StreamingContext  
)
```

### C# syntax

```
public override void GetObjectData(  
    SerializationInfo info,  
    StreamingContext context  
)
```

### Parameters

- **info** The `SerializationInfo` that holds the serialized object data about the exception being thrown.
- **context** The `StreamingContext` that contains contextual information about the source or destination.

### Remarks

Overrides `Exception.GetObjectData`.

## Errors property

Returns a collection of one or more `SAError` objects.

### Visual Basic syntax

```
Public ReadOnly Property Errors As SAErrorCollection
```

### C# syntax

```
public SAErrorCollection Errors {get;}
```

### Remarks

The `SAErrorCollection` object always contains at least one instance of the `SAError` object.

### See also

- [“SAErrorCollection class \[SQL Anywhere .NET\]” on page 244](#)
- [“SAError class \[SQL Anywhere .NET\]” on page 242](#)

## Message property

Returns the text describing the error.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Message As String
```

### C# syntax

```
public override string Message {get;}
```

### Remarks

This method returns a single string that contains a concatenation of all of the Message properties of all of the SAError objects in the Errors collection. Each message, except the last one, is followed by a carriage return.

### See also

- [“SAError class \[SQL Anywhere .NET\]” on page 242](#)

## NativeError property

Returns database-specific error information.

### Visual Basic syntax

```
Public ReadOnly Property NativeError As Integer
```

### C# syntax

```
public int NativeError {get;}
```

## Source property

Returns the name of the provider that generated the error.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Source As String
```

### C# syntax

```
public override string Source {get;}
```

## SAFactory class

Represents a set of methods for creating instances of the iAnywhere.Data.SQLAnywhere provider's implementation of the data source classes.

### Visual Basic syntax

```
Public NotInheritable Class SAFactory  
    Inherits System.Data.Common.DbProviderFactory
```

### C# syntax

```
public sealed class SAFactory : System.Data.Common.DbProviderFactory
```

### Base classes

- [System.Data.Common.DbProviderFactory](#)

### Members

All members of SAFactory class, including all inherited members.

Name	Description
<a href="#">CreateCommand method</a>	Returns a strongly typed System.Data.Common.DbCommand instance.
<a href="#">CreateCommandBuilder method</a>	Returns a strongly typed System.Data.Common.DbCommandBuilder instance.
<a href="#">CreateConnection method</a>	Returns a strongly typed System.Data.Common.DbConnection instance.
<a href="#">CreateConnectionStringBuilder method</a>	Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.
<a href="#">CreateDataAdapter method</a>	Returns a strongly typed System.Data.Common.DbDataAdapter instance.
<a href="#">CreateDataSourceEnumerator method</a>	Returns a strongly typed System.Data.Common.DbDataSourceEnumerator instance.
<a href="#">CreateParameter method</a>	Returns a strongly typed System.Data.Common.DbParameter instance.
<a href="#">CreatePermission method</a>	Returns a strongly-typed CodeAccessPermission instance.
<a href="#">CanCreateDataSourceEnumerator property</a>	Always returns true, which indicates that an SADataSourceEnumerator object can be created.
<a href="#">Instance field</a>	Represents the singleton instance of the SAFactory class.

### Remarks

There is no constructor for SAFactory.

ADO.NET 2.0 adds two new classes, `DbProviderFactories` and `DbProviderFactory`, to make provider independent code easier to write. To use them with SQL Anywhere specify `iAnywhere.Data.SQLAnywhere` as the provider invariant name passed to `GetFactory`. For example:

```
' Visual Basic
Dim factory As DbProviderFactory = _
    DbProviderFactories.GetFactory( "iAnywhere.Data.SQLAnywhere" )
Dim conn As DbConnection = _
    factory.CreateConnection()

// C#
DbProviderFactory factory =
    DbProviderFactories.GetFactory("iAnywhere.Data.SQLAnywhere" );
DbConnection conn = factory.CreateConnection();
```

In this example, `conn` is created as an `SACConnection` object.

For an explanation of provider factories and generic programming in ADO.NET 2.0, see <http://msdn2.microsoft.com/en-us/library/ms379620.aspx>.

The `SAFactory` class is not available in the .NET Compact Framework 2.0.

## CreateCommand method

Returns a strongly typed `System.Data.Common.DbCommand` instance.

### Visual Basic syntax

```
Public Overrides Function CreateCommand() As DbCommand
```

### C# syntax

```
public override DbCommand CreateCommand()
```

### Returns

A new `SACCommand` object typed as `DbCommand`.

### See also

- “[SACCommand class \[SQL Anywhere .NET\]](#)” on page 122

## CreateCommandBuilder method

Returns a strongly typed `System.Data.Common.DbCommandBuilder` instance.

### Visual Basic syntax

```
Public Overrides Function CreateCommandBuilder() As DbCommandBuilder
```

### C# syntax

```
public override DbCommandBuilder CreateCommandBuilder()
```

### Returns

A new SACommand object typed as DbCommand.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreateConnection method

Returns a strongly typed System.Data.Common.DbConnection instance.

### Visual Basic syntax

```
Public Overrides Function CreateConnection() As DbConnection
```

### C# syntax

```
public override DbConnection CreateConnection()
```

### Returns

A new SACommand object typed as DbCommand.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreateConnectionStringBuilder method

Returns a strongly typed System.Data.Common.DbConnectionStringBuilder instance.

### Visual Basic syntax

```
Public Overrides Function CreateConnectionStringBuilder()  
As DbConnectionStringBuilder
```

### C# syntax

```
public override DbConnectionStringBuilder CreateConnectionStringBuilder()
```

### Returns

A new SACommand object typed as DbCommand.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreateDataAdapter method

Returns a strongly typed System.Data.Common.DbDataAdapter instance.

### Visual Basic syntax

```
Public Overrides Function CreateDataAdapter() As DbDataAdapter
```

### C# syntax

```
public override DbDataAdapter CreateDataAdapter()
```

### Returns

A new SACommand object typed as DbCommand.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreateDataSourceEnumerator method

Returns a strongly typed System.Data.Common.DbDataSourceEnumerator instance.

### Visual Basic syntax

```
Public Overrides Function CreateDataSourceEnumerator()  
    As DbDataSourceEnumerator
```

### C# syntax

```
public override DbDataSourceEnumerator CreateDataSourceEnumerator()
```

### Returns

A new SACommand object typed as DbCommand.

### See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreateParameter method

Returns a strongly typed System.Data.Common.DbParameter instance.

### Visual Basic syntax

```
Public Overrides Function CreateParameter() As DbParameter
```

### C# syntax

```
public override DbParameter CreateParameter()
```

## Returns

A new SACommand object typed as DbCommand.

## See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CreatePermission method

Returns a strongly-typed CodeAccessPermission instance.

### Visual Basic syntax

```
Public Overrides Function CreatePermission(  
    ByVal state As PermissionState  
) As CodeAccessPermission
```

### C# syntax

```
public override CodeAccessPermission CreatePermission(  
    PermissionState state  
)
```

### Parameters

- **state** A member of the System.Security.Permissions.PermissionState enumeration.

## Returns

A new SACommand object typed as DbCommand.

## See also

- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## CanCreateDataSourceEnumerator property

Always returns true, which indicates that an SADataSourceEnumerator object can be created.

### Visual Basic syntax

```
Public ReadOnly Overrides Property CanCreateDataSourceEnumerator As  
Boolean
```

### C# syntax

```
public override bool CanCreateDataSourceEnumerator {get;}
```

## Returns

A new SACommand object typed as DbCommand.



**See also**

- [“SADataSourceEnumerator class \[SQL Anywhere .NET\]” on page 240](#)
- [“SACommand class \[SQL Anywhere .NET\]” on page 122](#)

## Instance field

Represents the singleton instance of the SAFactory class.

**Visual Basic syntax**

```
Public Shared ReadOnly Instance As SAFactory
```

**C# syntax**

```
public static readonly SAFactory Instance;
```

**Remarks**

SAFactory is a singleton class, which means only this instance of this class can exist.

Normally you would not use this field directly. Instead, you get a reference to this instance of SAFactory using `System.Data.Common.DbProviderFactories.GetFactory(String)`. For an example, see the SAFactory description.

The SAFactory class is not available in the .NET Compact Framework 2.0.

**See also**

- [“SAFactory class \[SQL Anywhere .NET\]” on page 249](#)

## SAInfoMessageEventArgs class

Provides data for the InfoMessage event.

**Visual Basic syntax**

```
Public NotInheritable Class SAInfoMessageEventArgs  
    Inherits System.EventArgs
```

**C# syntax**

```
public sealed class SAInfoMessageEventArgs : System.EventArgs
```

**Base classes**

- [System.EventArgs](#)

**Members**

All members of SAInfoMessageEventArgs class, including all inherited members.

Name	Description
<a href="#">ToString method</a>	Retrieves a string representation of the InfoMessage event.
<a href="#">Errors property</a>	Returns the collection of messages sent from the data source.
<a href="#">Message property</a>	Returns the full text of the error sent from the data source.
<a href="#">MessageType property</a>	Returns the type of the message.
<a href="#">NativeError property</a>	Returns the SQLCODE returned by the database.
<a href="#">Source property</a>	Returns the name of the SQL Anywhere .NET Data Provider.
<a href="#">Empty field</a> (Inherited from System.EventArgs)	Represents an event with no event data.

### Remarks

There is no constructor for SAInfoMessageEventArgs.

## ToString method

Retrieves a string representation of the InfoMessage event.

### Visual Basic syntax

```
Public Overrides Function ToString() As String
```

### C# syntax

```
public override string ToString()
```

### Returns

A string representing the InfoMessage event.

## Errors property

Returns the collection of messages sent from the data source.

### Visual Basic syntax

```
Public ReadOnly Property Errors As SAErrorCollection
```

### C# syntax

```
public SAErrorCollection Errors {get;}
```

## Message property

Returns the full text of the error sent from the data source.

### Visual Basic syntax

```
Public ReadOnly Property Message As String
```

### C# syntax

```
public string Message {get;}
```

## MessageType property

Returns the type of the message.

### Visual Basic syntax

```
Public ReadOnly Property MessageType As SAMessageType
```

### C# syntax

```
public SAMessageType MessageType {get;}
```

### Remarks

This can be one of: Action, Info, Status, or Warning.

## NativeError property

Returns the SQLCODE returned by the database.

### Visual Basic syntax

```
Public ReadOnly Property NativeError As Integer
```

### C# syntax

```
public int NativeError {get;}
```

## Source property

Returns the name of the SQL Anywhere .NET Data Provider.

### Visual Basic syntax

```
Public ReadOnly Property Source As String
```

### C# syntax

```
public string Source {get;}
```

## SAMetaDataCollectionNames class

Provides a list of constants for use with the `SACollection.GetSchema(string)` method to retrieve metadata collections.

### Visual Basic syntax

```
Public NotInheritable Class SAMetaDataCollectionNames
```

### C# syntax

```
public sealed class SAMetaDataCollectionNames
```

### Members

All members of SAMetaDataCollectionNames class, including all inherited members.

Name	Description
<a href="#">Columns field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the Columns collection.
<a href="#">DataSourceInformation field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the DataSourceInformation collection.
<a href="#">DataTypes field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the DataTypes collection.
<a href="#">ForeignKeys field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the ForeignKeys collection.
<a href="#">IndexColumns field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the IndexColumns collection.
<a href="#">Indexes field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the Indexes collection.
<a href="#">MetadataCollections field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the MetadataCollections collection.
<a href="#">ProcedureParameters field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the ProcedureParameters collection.
<a href="#">Procedures field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the Procedures collection.
<a href="#">ReservedWords field</a>	Provides a constant for use with the <code>SACollection.GetSchema(string)</code> method that represents the ReservedWords collection.

Name	Description
<a href="#">Restrictions field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the Restrictions collection.
<a href="#">Tables field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the Tables collection.
<a href="#">UserDefined-Types field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the UserDefinedTypes collection.
<a href="#">Users field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the Users collection.
<a href="#">ViewColumns field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the ViewColumns collection.
<a href="#">Views field</a>	Provides a constant for use with the <code>SACConnection.GetSchema(string)</code> method that represents the Views collection.

**Remarks**

This field is constant and read-only.

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

## Columns field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Columns collection.

**Visual Basic syntax**

```
Public Shared ReadOnly Columns As String
```

**C# syntax**

```
public static readonly string Columns;
```

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

**Example**

The following code fills a `DataTable` with the Columns collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Columns );
```

## DataSourceInformation field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `DataSourceInformation` collection.

### Visual Basic syntax

```
Public Shared ReadOnly DataSourceInformation As String
```

### C# syntax

```
public static readonly string DataSourceInformation;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the `DataSourceInformation` collection.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.DataSourceInformation );
```

## DataTypes field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `DataTypes` collection.

### Visual Basic syntax

```
Public Shared ReadOnly DataTypes As String
```

### C# syntax

```
public static readonly string DataTypes;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the `DataTypes` collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.DataTypes );
```

## ForeignKeys field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the `ForeignKeys` collection.

**Visual Basic syntax**

```
Public Shared ReadOnly ForeignKeys As String
```

**C# syntax**

```
public static readonly string ForeignKeys;
```

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

**Example**

The following code fills a DataTable with the ForeignKeys collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ForeignKeys );
```

## IndexColumns field

Provides a constant for use with the SACConnection.GetSchema(string) method that represents the IndexColumns collection.

**Visual Basic syntax**

```
Public Shared ReadOnly IndexColumns As String
```

**C# syntax**

```
public static readonly string IndexColumns;
```

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

**Example**

The following code fills a DataTable with the IndexColumns collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.IndexColumns );
```

## Indexes field

Provides a constant for use with the SACConnection.GetSchema(string) method that represents the Indexes collection.

**Visual Basic syntax**

```
Public Shared ReadOnly Indexes As String
```

**C# syntax**

```
public static readonly string Indexes;
```

### See also

- [“SAConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a DataTable with the Indexes collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Indexes );
```

## MetaDataCollections field

Provides a constant for use with the SAConnection.GetSchema(string) method that represents the MetaDataCollections collection.

### Visual Basic syntax

```
Public Shared ReadOnly MetaDataCollections As String
```

### C# syntax

```
public static readonly string MetaDataCollections;
```

### See also

- [“SAConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a DataTable with the MetaDataCollections collection.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.MetaDataCollections );
```

## ProcedureParameters field

Provides a constant for use with the SAConnection.GetSchema(string) method that represents the ProcedureParameters collection.

### Visual Basic syntax

```
Public Shared ReadOnly ProcedureParameters As String
```

### C# syntax

```
public static readonly string ProcedureParameters;
```

### See also

- [“SAConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a DataTable with the ProcedureParameters collection.



```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.ProcedureParameters );
```

## Procedures field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Procedures collection.

### Visual Basic syntax

```
Public Shared ReadOnly Procedures As String
```

### C# syntax

```
public static readonly string Procedures;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the Procedures collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Procedures );
```

## ReservedWords field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the ReservedWords collection.

### Visual Basic syntax

```
Public Shared ReadOnly ReservedWords As String
```

### C# syntax

```
public static readonly string ReservedWords;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the ReservedWords collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ReservedWords );
```

## Restrictions field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Restrictions collection.

### Visual Basic syntax

```
Public Shared ReadOnly Restrictions As String
```

### C# syntax

```
public static readonly string Restrictions;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the Restrictions collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Restrictions );
```

## Tables field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the Tables collection.

### Visual Basic syntax

```
Public Shared ReadOnly Tables As String
```

### C# syntax

```
public static readonly string Tables;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a `DataTable` with the Tables collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Tables );
```

## UserDefinedTypes field

Provides a constant for use with the `SACConnection.GetSchema(string)` method that represents the UserDefinedTypes collection.

**Visual Basic syntax**

```
Public Shared ReadOnly UserDefinedTypes As String
```

**C# syntax**

```
public static readonly string UserDefinedTypes;
```

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

**Example**

The following code fills a DataTable with the Users collection.

```
DataTable schema =  
    GetSchema( SAMetaDataCollectionNames.UserDefinedTypes );
```

## Users field

Provides a constant for use with the SACConnection.GetSchema(string) method that represents the Users collection.

**Visual Basic syntax**

```
Public Shared ReadOnly Users As String
```

**C# syntax**

```
public static readonly string Users;
```

**See also**

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

**Example**

The following code fills a DataTable with the Users collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Users );
```

## ViewColumns field

Provides a constant for use with the SACConnection.GetSchema(string) method that represents the ViewColumns collection.

**Visual Basic syntax**

```
Public Shared ReadOnly ViewColumns As String
```

**C# syntax**

```
public static readonly string ViewColumns;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a DataTable with the ViewColumns collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.ViewColumns );
```

## Views field

Provides a constant for use with the SACConnection.GetSchema(string) method that represents the Views collection.

### Visual Basic syntax

```
Public Shared ReadOnly Views As String
```

### C# syntax

```
public static readonly string Views;
```

### See also

- [“SACConnection.GetSchema method \[SQL Anywhere .NET\]” on page 168](#)

### Example

The following code fills a DataTable with the Views collection.

```
DataTable schema = GetSchema( SAMetaDataCollectionNames.Views );
```

## SAParameter class

Represents a parameter to an SACCommand, and optionally, its mapping to a DataSet column.

### Visual Basic syntax

```
Public NotInheritable Class SAParameter  
    Inherits System.Data.Common.DbParameter  
    Implements System.ICloneable
```

### C# syntax

```
public sealed class SAParameter :  
    System.Data.Common.DbParameter,  
    System.ICloneable
```

### Base classes

- [System.Data.Common.DbParameter](#)
- [System.ICloneable](#)

## Members

All members of SAParameter class, including all inherited members.

Name	Description
<a href="#">SAParameter constructor</a>	Initializes an SAParameter object with null (Nothing in Visual Basic) as its value.
<a href="#">ResetDbType method</a>	Resets the type (the values of DbType and SADBType) associated with this SAParameter.
<a href="#">ToString method</a>	Returns a string containing the ParameterName.
<a href="#">DbType property</a>	Gets and sets the DbType of the parameter.
<a href="#">Direction property</a>	Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
<a href="#">IsNullable property</a>	Gets and sets a value indicating whether the parameter accepts null values.
<a href="#">Offset property</a>	Gets and sets the offset to the Value property.
<a href="#">ParameterName property</a>	Gets and sets the name of the SAParameter.
<a href="#">Precision property</a>	Gets and sets the maximum number of digits used to represent the Value property.
<a href="#">SADBType property</a>	The SADBType of the parameter.
<a href="#">Scale property</a>	Gets and sets the number of decimal places to which Value is resolved.
<a href="#">Size property</a>	Gets and sets the maximum size, in bytes, of the data within the column.
<a href="#">SourceColumn property</a>	Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.
<a href="#">SourceColumnNullMapping property</a>	Gets and sets value that indicates whether the source column is nullable.
<a href="#">SourceVersion property</a>	Gets and sets the DataRowVersion to use when loading Value.
<a href="#">Value property</a>	Gets and sets the value of the parameter.

## Remarks

**Implements:** IDbDataParameter, IDataParameter, ICloneable

## SAParameter constructor

Initializes an SAParameter object with null (Nothing in Visual Basic) as its value.

### Overload list

Name	Description
<a href="#">SAParameter() constructor</a>	Initializes an SAParameter object with null (Nothing in Visual Basic) as its value.
<a href="#">SAParameter(string, object) constructor</a>	Initializes an SAParameter object with the specified parameter name and value.
<a href="#">SAParameter(string, SADbType) constructor</a>	Initializes an SAParameter object with the specified parameter name and data type.
<a href="#">SAParameter(string, SADbType, int) constructor</a>	Initializes an SAParameter object with the specified parameter name and data type.
<a href="#">SAParameter(string, SADbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor</a>	Initializes an SAParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.
<a href="#">SAParameter(string, SADbType, int, string) constructor</a>	Initializes an SAParameter object with the specified parameter name, data type, and length.

### SAParameter() constructor

Initializes an SAParameter object with null (Nothing in Visual Basic) as its value.

#### Visual Basic syntax

```
Public Sub New()
```

#### C# syntax

```
public SAParameter()
```

### SAParameter(string, object) constructor

Initializes an SAParameter object with the specified parameter name and value.

#### Visual Basic syntax

```
Public Sub New(ByVal parameterName As String, ByVal value As Object)
```

#### C# syntax

```
public SAParameter(string parameterName, object value)
```

**Parameters**

- **parameterName** The name of the parameter.
- **value** An Object that is the value of the parameter.

**Remarks**

This constructor is not recommended; it is provided for compatibility with other data providers.

**SAParameter(string, SADBType) constructor**

Initializes an SAParameter object with the specified parameter name and data type.

**Visual Basic syntax**

```
Public Sub New(ByVal parameterName As String, ByVal dbType As SADBType)
```

**C# syntax**

```
public SAParameter(string parameterName, SADBType dbType)
```

**Parameters**

- **parameterName** The name of the parameter.
- **dbType** One of the SADBType values.

**See also**

- [“SAParameter.SADBType property \[SQL Anywhere .NET\]” on page 274](#)

**SAParameter(string, SADBType, int) constructor**

Initializes an SAParameter object with the specified parameter name and data type.

**Visual Basic syntax**

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADBType,  
    ByVal size As Integer  
)
```

**C# syntax**

```
public SAParameter(string parameterName, SADBType dbType, int size)
```

**Parameters**

- **parameterName** The name of the parameter.
- **dbType** One of the SADBType values
- **size** The length of the parameter.

## SAParameter(string, SADbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object) constructor

Initializes an SAParameter object with the specified parameter name, data type, length, direction, nullability, numeric precision, numeric scale, source column, source version, and value.

### Visual Basic syntax

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADbType,  
    ByVal size As Integer,  
    ByVal direction As ParameterDirection,  
    ByVal isNullable As Boolean,  
    ByVal precision As Byte,  
    ByVal scale As Byte,  
    ByVal sourceColumn As String,  
    ByVal sourceVersion As DataRowVersion,  
    ByVal value As Object  
)
```

### C# syntax

```
public SAParameter(  
    string parameterName,  
    SADbType dbType,  
    int size,  
    ParameterDirection direction,  
    bool isNullable,  
    byte precision,  
    byte scale,  
    string sourceColumn,  
    DataRowVersion sourceVersion,  
    object value  
)
```

### Parameters

- **parameterName** The name of the parameter.
- **dbType** One of the SADbType values
- **size** The length of the parameter.
- **direction** One of the ParameterDirection values.
- **isNullable** True if the value of the field can be null; otherwise, false.
- **precision** The total number of digits to the left and right of the decimal point to which Value is resolved.
- **scale** The total number of decimal places to which Value is resolved.
- **sourceColumn** The name of the source column to map.
- **sourceVersion** One of the DataRowVersion values.



- **value** An Object that is the value of the parameter.

## SAParameter(string, SADbType, int, string) constructor

Initializes an SAParameter object with the specified parameter name, data type, and length.

### Visual Basic syntax

```
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADbType,  
    ByVal size As Integer,  
    ByVal sourceColumn As String  
)
```

### C# syntax

```
public SAParameter(  
    string parameterName,  
    SADbType dbType,  
    int size,  
    string sourceColumn  
)
```

### Parameters

- **parameterName** The name of the parameter.
- **dbType** One of the SADbType values
- **size** The length of the parameter.
- **sourceColumn** The name of the source column to map.

## ResetDbType method

Resets the type (the values of DbType and SADbType) associated with this SAParameter.

### Visual Basic syntax

```
Public Overrides Sub ResetDbType()
```

### C# syntax

```
public override void ResetDbType()
```

## ToString method

Returns a string containing the ParameterName.

### Visual Basic syntax

```
Public Overrides Function ToString() As String
```

### C# syntax

```
public override string ToString()
```

### Returns

The name of the parameter.

## DbType property

Gets and sets the DbType of the parameter.

### Visual Basic syntax

```
Public Overrides Property DbType As DbType
```

### C# syntax

```
public override DbType DbType {get;set;}
```

### Remarks

The SADBType and DbType are linked. Therefore, setting the DbType changes the SADBType to a supporting SADBType.

The value must be a member of the SADBType enumerator.

## Direction property

Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

### Visual Basic syntax

```
Public Overrides Property Direction As ParameterDirection
```

### C# syntax

```
public override ParameterDirection Direction {get;set;}
```

### Remarks

One of the ParameterDirection values.

If the ParameterDirection is output, and execution of the associated SACommand does not return a value, the SAParameter contains a null value. After the last row from the last result set is read, the Output, InputOut, and ReturnValue parameters are updated.

## IsNullable property

Gets and sets a value indicating whether the parameter accepts null values.

### Visual Basic syntax

```
Public Overrides Property IsNullable As Boolean
```

### C# syntax

```
public override bool IsNullable {get;set;}
```

### Remarks

This property is true if null values are accepted; otherwise, it is false. The default is false. Null values are handled using the DBNull class.

## Offset property

Gets and sets the offset to the Value property.

### Visual Basic syntax

```
Public Property Offset As Integer
```

### C# syntax

```
public int Offset {get;set;}
```

### Remarks

The offset to the value. The default is 0.

## ParameterName property

Gets and sets the name of the SAParameter.

### Visual Basic syntax

```
Public Overrides Property ParameterName As String
```

### C# syntax

```
public override string ParameterName {get;set;}
```

### Remarks

The default is an empty string.

The SQL Anywhere .NET Data Provider uses positional parameters that are marked with a question mark (?) instead of named parameters.

## Precision property

Gets and sets the maximum number of digits used to represent the Value property.

### Visual Basic syntax

```
Public Property Precision As Byte
```

### C# syntax

```
public byte Precision {get;set;}
```

### Remarks

The value of this property is the maximum number of digits used to represent the Value property. The default value is 0, which indicates that the data provider sets the precision for the Value property.

The Precision property is only used for decimal and numeric input parameters.

## SADbType property

The SADbType of the parameter.

### Visual Basic syntax

```
Public Property SADbType As SADbType
```

### C# syntax

```
public SADbType SADbType {get;set;}
```

### Remarks

The SADbType and DbType are linked. Therefore, setting the SADbType changes the DbType to a supporting DbType.

The value must be a member of the SADbType enumerator.

## Scale property

Gets and sets the number of decimal places to which Value is resolved.

### Visual Basic syntax

```
Public Property Scale As Byte
```

### C# syntax

```
public byte Scale {get;set;}
```

### Remarks

The number of decimal places to which Value is resolved. The default is 0.

The Scale property is only used for decimal and numeric input parameters.

## Size property

Gets and sets the maximum size, in bytes, of the data within the column.

### Visual Basic syntax

```
Public Overrides Property Size As Integer
```

### C# syntax

```
public override int Size {get;set;}
```

### Remarks

The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.

The Size property is used for binary and string types.

For variable length data types, the Size property describes the maximum amount of data to transmit to the server. For example, the Size property can be used to limit the amount of data sent to the server for a string value to the first one hundred bytes.

If not explicitly set, the size is inferred from the actual size of the specified parameter value. For fixed width data types, the value of Size is ignored. It can be retrieved for informational purposes, and returns the maximum amount of bytes the provider uses when transmitting the value of the parameter to the server.

## SourceColumn property

Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.

### Visual Basic syntax

```
Public Overrides Property SourceColumn As String
```

### C# syntax

```
public override string SourceColumn {get;set;}
```

### Remarks

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

When `SourceColumn` is set to anything other than an empty string, the value of the parameter is retrieved from the column with the `SourceColumn` name. If `Direction` is set to `Input`, the value is taken from the `DataSet`. If `Direction` is set to `Output`, the value is taken from the data source. A `Direction` of `InputOutput` is a combination of both.

## SourceColumnNullMapping property

Gets and sets value that indicates whether the source column is nullable.

### Visual Basic syntax

```
Public Overrides Property SourceColumnNullMapping As Boolean
```

### C# syntax

```
public override bool SourceColumnNullMapping {get;set;}
```

### Remarks

This allows `SACommandBuilder` to generate `Update` statements for nullable columns correctly.

If the source column is nullable, `true` is returned; otherwise, `false`.

## SourceVersion property

Gets and sets the `DataRowVersion` to use when loading `Value`.

### Visual Basic syntax

```
Public Overrides Property SourceVersion As DataRowVersion
```

### C# syntax

```
public override DataRowVersion SourceVersion {get;set;}
```

### Remarks

Used by `UpdateCommand` during an `Update` operation to determine whether the parameter value is set to `Current` or `Original`. This allows primary keys to be updated. This property is ignored by `InsertCommand` and `DeleteCommand`. This property is set to the version of the `DataRow` used by the `Item` property, or the `GetChildRows` method of the `DataRow` object.

## Value property

Gets and sets the value of the parameter.

### Visual Basic syntax

```
Public Overrides Property Value As Object
```

**C# syntax**

```
public override object value {get;set;}
```

**Remarks**

An Object that specifies the value of the parameter.

For input parameters, the value is bound to the SACommand that is sent to the server. For output and return value parameters, the value is set on completion of the SACommand and after the SADataReader is closed.

When sending a null parameter value to the server, you must specify DBNull, not null. The null value in the system is an empty object that has no value. DBNull is used to represent null values.

If the application specifies the database type, the bound value is converted to that type when the SQL Anywhere .NET Data Provider sends the data to the server. The provider attempts to convert any type of value if it supports the IConvertible interface. Conversion errors may result if the specified type is not compatible with the value.

Both the DbType and SADbType properties can be inferred by setting the Value.

The Value property is overwritten by Update.

## SAParameterCollection class

Represents all parameters to an SACommand object and, optionally, their mapping to a DataSet column.

**Visual Basic syntax**

```
Public NotInheritable Class SAParameterCollection
    Inherits System.Data.Common.DbParameterCollection
```

**C# syntax**

```
public sealed class SAParameterCollection :
    System.Data.Common.DbParameterCollection
```

**Base classes**

- [System.Data.Common.DbParameterCollection](#)

**Members**

All members of SAParameterCollection class, including all inherited members.

Name	Description
<a href="#">Add method</a>	Adds an SAParameter object to this collection.
<a href="#">AddRange method</a>	Adds an array of values to the end of the SAParameterCollection.

Name	Description
<a href="#">AddWithValue method</a>	Adds a value to the end of this collection.
<a href="#">Clear method</a>	Removes all items from the collection.
<a href="#">Contains method</a>	Indicates whether an SAParameter object exists in the collection.
<a href="#">CopyTo method</a>	Copies SAParameter objects from the SAParameterCollection to the specified array.
<a href="#">GetEnumerator method</a>	Returns an enumerator that iterates through the SAParameterCollection.
<a href="#">GetParameter method</a> (Inherited from System.Data.Common.DbParameterCollection)	Returns the <a href="#">System.Data.Common.DbParameter</a> object at the specified index in the collection.
<a href="#">IndexOf method</a>	Returns the location of the SAParameter object in the collection.
<a href="#">Insert method</a>	Inserts an SAParameter object in the collection at the specified index.
<a href="#">Remove method</a>	Removes the specified SAParameter object from the collection.
<a href="#">RemoveAt method</a>	Removes the specified SAParameter object from the collection.
<a href="#">SetParameter method</a> (Inherited from System.Data.Common.DbParameterCollection)	Sets the <a href="#">System.Data.Common.DbParameter</a> object at the specified index to a new value.
<a href="#">Count property</a>	Returns the number of SAParameter objects in the collection.
<a href="#">IsFixedSize property</a>	Gets a value that indicates whether the SAParameterCollection has a fixed size.
<a href="#">IsReadOnly property</a>	Gets a value that indicates whether the SAParameterCollection is read-only.
<a href="#">IsSynchronized property</a>	Gets a value that indicates whether the SAParameterCollection object is synchronized.
<a href="#">SyncRoot property</a>	Gets an object that can be used to synchronize access to the SAParameterCollection.
<a href="#">this property</a>	Gets and sets the SAParameter object at the specified index.

**Remarks**

There is no constructor for SAParameterCollection. You obtain an SAParameterCollection object from the SACommand.Parameters property of an SACommand object.



**See also**

- “SACommand class [SQL Anywhere .NET]” on page 122
- “SACommand.Parameters property [SQL Anywhere .NET]” on page 143
- “SAParameter class [SQL Anywhere .NET]” on page 266
- “SAParameterCollection class [SQL Anywhere .NET]” on page 277

**Add method**

Adds an SAParameter object to this collection.

**Overload list**

Name	Description
<a href="#">Add(object) method</a>	Adds an SAParameter object to this collection.
<a href="#">Add(SAParameter) method</a>	Adds an SAParameter object to this collection.
<a href="#">Add(string, object) method</a>	Adds an SAParameter object to this collection, created using the specified parameter name and value, to the collection.
<a href="#">Add(string, SADB-Type) method</a>	Adds an SAParameter object to this collection, created using the specified parameter name and data type, to the collection.
<a href="#">Add(string, SADB-Type, int) method</a>	Adds an SAParameter object to this collection, created using the specified parameter name, data type, and length, to the collection.
<a href="#">Add(string, SADB-Type, int, string) method</a>	Adds an SAParameter object to this collection, created using the specified parameter name, data type, length, and source column name, to the collection.

**Add(object) method**

Adds an SAParameter object to this collection.

**Visual Basic syntax**

```
Public Overrides Function Add(ByVal value As Object) As Integer
```

**C# syntax**

```
public override int Add(object value)
```

**Parameters**

- **value** The SAParameter object to add to the collection.

## Returns

The index of the new SAParameter object.

## See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)

## Add(SAParameter) method

Adds an SAParameter object to this collection.

### Visual Basic syntax

```
Public Function Add(ByVal value As SAParameter) As SAParameter
```

### C# syntax

```
public SAParameter Add(SAParameter value)
```

### Parameters

- **value** The SAParameter object to add to the collection.

### Returns

The new SAParameter object.

## Add(string, object) method

Adds an SAParameter object to this collection, created using the specified parameter name and value, to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal value As Object  
) As SAParameter
```

### C# syntax

```
public SAParameter Add(string parameterName, object value)
```

### Parameters

- **parameterName** The name of the parameter.
- **value** The value of the parameter to add to the connection.

### Returns

The new SAParameter object.

## Remarks

Because of the special treatment of the 0 and 0.0 constants and the way overloaded methods are resolved, it is highly recommended that you explicitly cast constant values to type object when using this method.

## See also

- [“SAPParameter class \[SQL Anywhere .NET\]” on page 266](#)

## Add(string, SADBType) method

Adds an SAPParameter object to this collection, created using the specified parameter name and data type, to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal saDbType As SADBType  
) As SAPParameter
```

### C# syntax

```
public SAPParameter Add(string parameterName, SADBType saDbType)
```

### Parameters

- **parameterName** The name of the parameter.
- **saDbType** One of the SADBType values.

### Returns

The new SAPParameter object.

### See also

- [“SADBType enumeration \[SQL Anywhere .NET\]” on page 319](#)
- [“SAPParameterCollection.Add method \[SQL Anywhere .NET\]” on page 279](#)

## Add(string, SADBType, int) method

Adds an SAPParameter object to this collection, created using the specified parameter name, data type, and length, to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal saDbType As SADBType,  
    ByVal size As Integer  
) As SAPParameter
```

### C# syntax

```
public SAPParameter Add(  
    string parameterName,  
    SADBType saDbType,  
    int size  
)
```

### Parameters

- **parameterName** The name of the parameter.
- **saDbType** One of the SADBType values.
- **size** The length of the parameter.

### Returns

The new SAPParameter object.

### See also

- [“SADBType enumeration \[SQL Anywhere .NET\]” on page 319](#)
- [“SAPParameterCollection.Add method \[SQL Anywhere .NET\]” on page 279](#)

## Add(string, SADBType, int, string) method

Adds an SAPParameter object to this collection, created using the specified parameter name, data type, length, and source column name, to the collection.

### Visual Basic syntax

```
Public Function Add(  
    ByVal parameterName As String,  
    ByVal saDbType As SADBType,  
    ByVal size As Integer,  
    ByVal sourceColumn As String  
) As SAPParameter
```

### C# syntax

```
public SAPParameter Add(  
    string parameterName,  
    SADBType saDbType,  
    int size,  
    string sourceColumn  
)
```

### Parameters

- **parameterName** The name of the parameter.
- **saDbType** One of the SADBType values.
- **size** The length of the column.
- **sourceColumn** The name of the source column to map.

**Returns**

The new SAParameter object.

**See also**

- [“SADBType enumeration \[SQL Anywhere .NET\]” on page 319](#)
- [“SAParameterCollection.Add method \[SQL Anywhere .NET\]” on page 279](#)

## AddRange method

Adds an array of values to the end of the SAParameterCollection.

**Overload list**

Name	Description
<a href="#">AddRange(Array) method</a>	Adds an array of values to the end of the SAParameterCollection.
<a href="#">AddRange(SAParameter[]) method</a>	Adds an array of values to the end of the SAParameterCollection.

### AddRange(Array) method

Adds an array of values to the end of the SAParameterCollection.

**Visual Basic syntax**

```
Public Overrides Sub AddRange(ByVal values As Array)
```

**C# syntax**

```
public override void AddRange(Array values)
```

**Parameters**

- **values** The values to add.

### AddRange(SAParameter[]) method

Adds an array of values to the end of the SAParameterCollection.

**Visual Basic syntax**

```
Public Sub AddRange(ByVal values As SAParameter())
```

**C# syntax**

```
public void AddRange(SAParameter[] values)
```

**Parameters**

- **values** An array of SAParameter objects to add to the end of this collection.

## AddWithValue method

Adds a value to the end of this collection.

### Visual Basic syntax

```
Public Function AddWithValue(  
    ByVal parameterName As String,  
    ByVal value As Object  
) As SAParameter
```

### C# syntax

```
public SAParameter AddWithValue(string parameterName, object value)
```

### Parameters

- **parameterName** The name of the parameter.
- **value** The value to be added.

### Returns

The new SAParameter object.

## Clear method

Removes all items from the collection.

### Visual Basic syntax

```
Public Overrides Sub Clear()
```

### C# syntax

```
public override void Clear()
```

## Contains method

Indicates whether an SAParameter object exists in the collection.

### Overload list

Name	Description
<a href="#">Contains(object) method</a>	Indicates whether an SAParameter object exists in the collection.
<a href="#">Contains(string) method</a>	Indicates whether an SAParameter object exists in the collection.

## Contains(object) method

Indicates whether an SAParameter object exists in the collection.

### Visual Basic syntax

```
Public Overrides Function Contains(ByVal value As Object) As Boolean
```

### C# syntax

```
public override bool Contains(object value)
```

### Parameters

- **value** The SAParameter object to find.

### Returns

True if the collection contains the SAParameter object. Otherwise, false.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection.Contains method \[SQL Anywhere .NET\]” on page 284](#)

## Contains(string) method

Indicates whether an SAParameter object exists in the collection.

### Visual Basic syntax

```
Public Overrides Function Contains(ByVal value As String) As Boolean
```

### C# syntax

```
public override bool Contains(string value)
```

### Parameters

- **value** The name of the parameter to search for.

### Returns

True if the collection contains the SAParameter object. Otherwise, false.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection.Contains method \[SQL Anywhere .NET\]” on page 284](#)

## CopyTo method

Copies SAParameter objects from the SAParameterCollection to the specified array.

### Visual Basic syntax

```
Public Overrides Sub CopyTo(  
    ByVal array As Array,  
    ByVal index As Integer  
)
```

### C# syntax

```
public override void CopyTo(Array array, int index)
```

### Parameters

- **array** The array to copy the SAParameter objects into.
- **index** The starting index of the array.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)

## GetEnumerator method

Returns an enumerator that iterates through the SAParameterCollection.

### Visual Basic syntax

```
Public Overrides Function GetEnumerator()  
    As System.Collections.IEnumerator
```

### C# syntax

```
public override IEnumerator GetEnumerator()
```

### Returns

An System.Collections.IEnumerator for the SAParameterCollection object.

### See also

- [“SAParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)

## IndexOf method

Returns the location of the SAParameter object in the collection.

### Overload list

Name	Description
<a href="#">IndexOf(object) method</a>	Returns the location of the SAParameter object in the collection.



Name	Description
<a href="#">IndexOf(string) method</a>	Returns the location of the SAParameter object in the collection.

## IndexOf(object) method

Returns the location of the SAParameter object in the collection.

### Visual Basic syntax

```
Public Overrides Function IndexOf(ByVal value As Object) As Integer
```

### C# syntax

```
public override int IndexOf(object value)
```

### Parameters

- **value** The SAParameter object to locate.

### Returns

The zero-based location of the SAParameter object in the collection.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection.IndexOf method \[SQL Anywhere .NET\]” on page 286](#)

## IndexOf(string) method

Returns the location of the SAParameter object in the collection.

### Visual Basic syntax

```
Public Overrides Function IndexOf(  
    ByVal parameterName As String  
) As Integer
```

### C# syntax

```
public override int IndexOf(string parameterName)
```

### Parameters

- **parameterName** The name of the parameter to locate.

### Returns

The zero-based index of the SAParameter object in the collection.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection.IndexOf method \[SQL Anywhere .NET\]” on page 286](#)

## Insert method

Inserts an SAParameter object in the collection at the specified index.

### Visual Basic syntax

```
Public Overrides Sub Insert(  
    ByVal index As Integer,  
    ByVal value As Object  
)
```

### C# syntax

```
public override void Insert(int index, object value)
```

### Parameters

- **index** The zero-based index where the parameter is to be inserted within the collection.
- **value** The SAParameter object to add to the collection.

## Remove method

Removes the specified SAParameter object from the collection.

### Visual Basic syntax

```
Public Overrides Sub Remove(ByVal value As Object)
```

### C# syntax

```
public override void Remove(object value)
```

### Parameters

- **value** The SAParameter object to remove from the collection.

## RemoveAt method

Removes the specified SAParameter object from the collection.

**Overload list**

Name	Description
<a href="#">RemoveAt(int) method</a>	Removes the specified SAParameter object from the collection.
<a href="#">RemoveAt(string) method</a>	Removes the specified SAParameter object from the collection.

**RemoveAt(int) method**

Removes the specified SAParameter object from the collection.

**Visual Basic syntax**

```
Public Overrides Sub RemoveAt(ByVal index As Integer)
```

**C# syntax**

```
public override void RemoveAt(int index)
```

**Parameters**

- **index** The zero-based index of the parameter to remove.

**See also**

- [“SAParameterCollection.RemoveAt method \[SQL Anywhere .NET\]” on page 288](#)

**RemoveAt(string) method**

Removes the specified SAParameter object from the collection.

**Visual Basic syntax**

```
Public Overrides Sub RemoveAt(ByVal parameterName As String)
```

**C# syntax**

```
public override void RemoveAt(string parameterName)
```

**Parameters**

- **parameterName** The name of the SAParameter object to remove.

**See also**

- [“SAParameterCollection.RemoveAt method \[SQL Anywhere .NET\]” on page 288](#)

**Count property**

Returns the number of SAParameter objects in the collection.

### Visual Basic syntax

```
Public ReadOnly Overrides Property Count As Integer
```

### C# syntax

```
public override int Count {get;}
```

### Remarks

The number of SAParameter objects in the collection.

### See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)

## IsFixedSize property

Gets a value that indicates whether the SAParameterCollection has a fixed size.

### Visual Basic syntax

```
Public ReadOnly Overrides Property IsFixedSize As Boolean
```

### C# syntax

```
public override bool IsFixedSize {get;}
```

### Remarks

True if this collection has a fixed size, false otherwise.

## IsReadOnly property

Gets a value that indicates whether the SAParameterCollection is read-only.

### Visual Basic syntax

```
Public ReadOnly Overrides Property IsReadOnly As Boolean
```

### C# syntax

```
public override bool IsReadOnly {get;}
```

### Remarks

True if this collection is read-only, false otherwise.

## IsSynchronized property

Gets a value that indicates whether the SAParameterCollection object is synchronized.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property IsSynchronized As Boolean
```

**C# syntax**

```
public override bool IsSynchronized {get;}
```

**Remarks**

True if this collection is synchronized, false otherwise.

## SyncRoot property

Gets an object that can be used to synchronize access to the SAParameterCollection.

**Visual Basic syntax**

```
Public ReadOnly Overrides Property SyncRoot As Object
```

**C# syntax**

```
public override object SyncRoot {get;}
```

## this property

Gets and sets the SAParameter object at the specified index.

**Overload list**

Name	Description
<a href="#">this[int] property</a>	Gets and sets the SAParameter object at the specified index.
<a href="#">this[string] property</a>	Gets and sets the SAParameter object at the specified index.

## this[int] property

Gets and sets the SAParameter object at the specified index.

**Visual Basic syntax**

```
Public Shadows Property Item(ByVal index As Integer) As SAParameter
```

**C# syntax**

```
public new SAParameter this[int index] {get;set;}
```

**Parameters**

- **index** The zero-based index of the parameter to retrieve.

## Returns

The SAParameter at the specified index.

## Remarks

An SAParameter object.

In C#, this property is the indexer for the SAParameterCollection object.

## See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)

## this[string] property

Gets and sets the SAParameter object at the specified index.

## Visual Basic syntax

```
Public Shadows Property Item(  
    ByVal parameterName As String  
) As SAParameter
```

## C# syntax

```
public new SAParameter this[string parameterName] {get;set;}
```

## Parameters

- **parameterName** The name of the parameter to retrieve.

## Returns

The SAParameter object with the specified name.

## Remarks

An SAParameter object.

In C#, this property is the indexer for the SAParameterCollection object.

## See also

- [“SAParameter class \[SQL Anywhere .NET\]” on page 266](#)
- [“SAParameterCollection class \[SQL Anywhere .NET\]” on page 277](#)
- [“SADataReader.GetOrdinal method \[SQL Anywhere .NET\]” on page 228](#)
- [“SADataReader.GetValue method \[SQL Anywhere .NET\]” on page 233](#)
- [“SADataReader.GetFieldType method \[SQL Anywhere .NET\]” on page 225](#)

## SAPermission class

Enables the SQL Anywhere .NET Data Provider to ensure that a user has a security level adequate to access a SQL Anywhere data source.

### Visual Basic syntax

```
Public NotInheritable Class SAPermission
    Inherits System.Data.Common.DBDataPermission
```

### C# syntax

```
public sealed class SAPermission : System.Data.Common.DBDataPermission
```

### Base classes

- [System.Data.Common.DBDataPermission](#)

### Members

All members of SAPermission class, including all inherited members.

Name	Description
<a href="#">SAPermission constructor</a>	Initializes a new instance of the SAPermission class.
<a href="#">Add method</a> (Inherited from <a href="#">System.Data.Common.DBDataPermission</a> )	Adds access for the specified connection string to the existing state of the DBDataPermission.
<a href="#">Clear method</a> (Inherited from <a href="#">System.Data.Common.DBDataPermission</a> )	Removes all permissions that were previous added using the <a href="#">System.Data.Common.DBDataPermission.Clear(System.String,System.String,System.Data.KeyRestrictionBehavior)</a> method.

Name	Description
<a href="#">Copy method</a> (Inherited from System.Data.Common.DbDataPermission)	Creates and returns an identical copy of the current permission object.
<a href="#">FromXml method</a> (Inherited from System.Data.Common.DbDataPermission)	Reconstructs a security object with a specified state from an XML encoding.
<a href="#">Intersect method</a> (Inherited from System.Data.Common.DbDataPermission)	Returns a new permission object representing the intersection of the current permission object and the specified permission object.
<a href="#">IsSubsetOf method</a> (Inherited from System.Data.Common.DbDataPermission)	Returns a value indicating whether the current permission object is a subset of the specified permission object.



Name	Description
<a href="#">IsUnrestricted method</a> (Inherited from System.Data.Common.DBDataPermission)	Returns a value indicating whether the permission can be represented as unrestricted without any knowledge of permission semantics.
<a href="#">ToXml method</a> (Inherited from System.Data.Common.DBDataPermission)	Creates an XML encoding of the security object and its current state.
<a href="#">Union method</a> (Inherited from System.Data.Common.DBDataPermission)	Returns a new permission object that is the union of the current and specified permission objects.
<a href="#">AllowBlankPassword property</a> (Inherited from System.Data.Common.DBDataPermission)	Gets a value indicating whether a blank password is allowed.

## Remarks

**Base classes** DBDataPermission

## SAPermission constructor

Initializes a new instance of the SAPermission class.

### Visual Basic syntax

```
Public Sub New(ByVal state As PermissionState)
```

### C# syntax

```
public SAPermission(PermissionState state)
```

### Parameters

- **state** One of the PermissionState values.

## SAPermissionAttribute class

Associates a security action with a custom security attribute.

### Visual Basic syntax

```
Public NotInheritable Class SAPermissionAttribute  
    Inherits System.Data.Common.DBDataPermissionAttribute
```

### C# syntax

```
public sealed class SAPermissionAttribute :  
    System.Data.Common.DBDataPermissionAttribute
```

### Base classes

- [System.Data.Common.DBDataPermissionAttribute](#)

### Members

All members of SAPermissionAttribute class, including all inherited members.

Name	Description
<a href="#">SAPermissionAttribute constructor</a>	Initializes a new instance of the SAPermissionAttribute class.
<a href="#">CreatePermission method</a>	Returns an SAPermission object that is configured according to the attribute properties.
<a href="#">ShouldSerializeConnectionString method</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Identifies whether the attribute should serialize the connection string.

Name	Description
<a href="#">ShouldSerializeKeyRestrictions method</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Identifies whether the attribute should serialize the set of key restrictions.
<a href="#">AllowBlankPassword property</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Gets or sets a value indicating whether a blank password is allowed.
<a href="#">ConnectionString property</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Gets or sets a permitted connection string.
<a href="#">KeyRestrictionBehavior property</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Identifies whether the list of connection string parameters identified by the <a href="#">System.Data.Common.DBDataPermissionAttribute.KeyRestrictions</a> property are the only connection string parameters allowed.
<a href="#">KeyRestrictions property</a> (Inherited from System.Data.Common.DBDataPermissionAttribute)	Gets or sets connection string parameters that are allowed or disallowed.

## SAPermissionAttribute constructor

Initializes a new instance of the SAPermissionAttribute class.

### Visual Basic syntax

```
Public Sub New(ByVal action As SecurityAction)
```

### C# syntax

```
public SAPermissionAttribute(SecurityAction action)
```

### Parameters

- **action** One of the SecurityAction values representing an action that can be performed using declarative security.

### Returns

An SAPermissionAttribute object.

## CreatePermission method

Returns an SAPermission object that is configured according to the attribute properties.

### Visual Basic syntax

```
Public Overrides Function CreatePermission() As IPermission
```

### C# syntax

```
public override IPermission CreatePermission()
```

## SARowsCopiedEventArgs class

Represents the set of arguments passed to the SARowsCopiedEventHandler.

### Visual Basic syntax

```
Public NotInheritable Class SARowsCopiedEventArgs
```

### C# syntax

```
public sealed class SARowsCopiedEventArgs
```

### Members

All members of SARowsCopiedEventArgs class, including all inherited members.

Name	Description
<a href="#">SARowsCopiedEventArgs constructor</a>	Creates a new instance of the SARowsCopiedEventArgs object.
<a href="#">Abort property</a>	Gets or sets a value that indicates whether the bulk-copy operation should be aborted.
<a href="#">RowsCopied property</a>	Gets the number of rows copied during the current bulk-copy operation.

### Remarks

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

## SARowsCopiedEventArgs constructor

Creates a new instance of the SARowsCopiedEventArgs object.

### Visual Basic syntax

```
Public Sub New(ByVal rowsCopied As Long)
```

### C# syntax

```
public SARowsCopiedEventArgs(long rowsCopied)
```

**Parameters**

- **rowsCopied** An 64-bit integer value that indicates the number of rows copied during the current bulk-copy operation.

**Remarks**

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

## Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

**Visual Basic syntax**

```
Public Property Abort As Boolean
```

**C# syntax**

```
public bool Abort {get;set;}
```

**Remarks**

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

## RowsCopied property

Gets the number of rows copied during the current bulk-copy operation.

**Visual Basic syntax**

```
Public ReadOnly Property RowsCopied As Long
```

**C# syntax**

```
public long RowsCopied {get;}
```

**Remarks**

The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

## SARowUpdatedEventArgs class

Provides data for the RowUpdated event.

**Visual Basic syntax**

```
Public NotInheritable Class SARowUpdatedEventArgs  
    Inherits System.Data.Common.RowUpdatedEventArgs
```

### C# syntax

```
public sealed class SARowUpdatedEventArgs :
    System.Data.Common.RowUpdatedEventArgs
```

### Base classes

- [System.Data.Common.RowUpdatedEventArgs](#)

### Members

All members of SARowUpdatedEventArgs class, including all inherited members.

Name	Description
<a href="#">SARowUpdatedEventArgs constructor</a>	Initializes a new instance of the SARowUpdatedEventArgs class.
<a href="#">CopyToRows method</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Copies references to the modified rows into the provided array.
<a href="#">Command property</a>	Gets the SACommand that is executed when DataAdapter.Update is called.
<a href="#">Errors property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the <a href="#">System.Data.Common.RowUpdatedEventArgs.Command</a> was executed.
<a href="#">RecordsAffected property</a>	Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.
<a href="#">Row property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the <a href="#">System.Data.DataRow</a> sent through an <a href="#">System.Data.Common.DbDataAdapter.Update(System.Data.DataSet)</a> .
<a href="#">RowCount property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
<a href="#">StatementType property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the type of SQL statement executed.
<a href="#">Status property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the <a href="#">System.Data.UpdateStatus</a> of the <a href="#">System.Data.Common.RowUpdatedEventArgs.Command</a> property.
<a href="#">TableMapping property</a> (Inherited from System.Data.Common.RowUpdatedEventArgs)	Gets the <a href="#">System.Data.Common.DataTableMapping</a> sent through an <a href="#">System.Data.Common.DbDataAdapter.Update(System.Data.DataSet)</a> .

## SARowUpdatedEventArgs constructor

Initializes a new instance of the SARowUpdatedEventArgs class.

### Visual Basic syntax

```
Public Sub New(  
    ByVal row As DataRow,  
    ByVal command As IDbCommand,  
    ByVal statementType As StatementType,  
    ByVal tableMapping As DataTableMapping  
)
```

### C# syntax

```
public SARowUpdatedEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
)
```

### Parameters

- **row** The DataRow sent through an Update.
- **command** The IDbCommand executed when Update is called.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an Update.

## Command property

Gets the SACommand that is executed when DataAdapter.Update is called.

### Visual Basic syntax

```
Public ReadOnly Shadows Property Command As SACommand
```

### C# syntax

```
public new SACommand Command {get;}
```

## RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

### Visual Basic syntax

```
Public ReadOnly Shadows Property RecordsAffected As Integer
```

### C# syntax

```
public new int RecordsAffected {get;}
```

### Remarks

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.

## SARowUpdatingEventArgs class

Provides data for the RowUpdating event.

### Visual Basic syntax

```
Public NotInheritable Class SARowUpdatingEventArgs  
    Inherits System.Data.Common.RowUpdatingEventArgs
```

### C# syntax

```
public sealed class SARowUpdatingEventArgs :  
    System.Data.Common.RowUpdatingEventArgs
```

### Base classes

- [System.Data.Common.RowUpdatingEventArgs](#)

### Members

All members of SARowUpdatingEventArgs class, including all inherited members.

Name	Description
<a href="#">SARowUpdatingEventArgs constructor</a>	Initializes a new instance of the SARowUpdatingEventArgs class.
<a href="#">BaseCommand property</a> (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets or sets the <a href="#">System.Data.IDbCommand</a> object for an instance of this class.
<a href="#">Command property</a>	Specifies the SACommand to execute when performing the Update.
<a href="#">Errors property</a> (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the <a href="#">System.Data.Common.RowUpdatedEventArgs.Command</a> executes.
<a href="#">Row property</a> (Inherited from System.Data.Common.RowUpdatingEventArgs)	Gets the <a href="#">System.Data.DataRow</a> that will be sent to the server as part of an insert, update, or delete operation.



Name	Description
<a href="#">StatementType property</a> (Inherited from <a href="#">System.Data.Common.RowUpdatingEventArgs</a> )	Gets the type of SQL statement to execute.
<a href="#">Status property</a> (Inherited from <a href="#">System.Data.Common.RowUpdatingEventArgs</a> )	Gets or sets the <a href="#">System.Data.UpdateStatus</a> of the <a href="#">System.Data.Common.RowUpdatedEventArgs.Command</a> property.
<a href="#">TableMapping property</a> (Inherited from <a href="#">System.Data.Common.RowUpdatingEventArgs</a> )	Gets the <a href="#">System.Data.Common.DataTableMapping</a> to send through the <a href="#">System.Data.Common.DbDataAdapter.Update(System.Data.DataSet)</a> .

## SARowUpdatingEventArgs constructor

Initializes a new instance of the SARowUpdatingEventArgs class.

### Visual Basic syntax

```
Public Sub New(  
    ByVal row As DataRow,  
    ByVal command As IDbCommand,  
    ByVal statementType As StatementType,  
    ByVal tableMapping As DataTableMapping  
)
```

### C# syntax

```
public SARowUpdatingEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
)
```

### Parameters

- **row** The DataRow to update.
- **command** The IDbCommand to execute during update.
- **statementType** One of the StatementType values that specifies the type of query executed.
- **tableMapping** The DataTableMapping sent through an Update.

## Command property

Specifies the SACommand to execute when performing the Update.

### Visual Basic syntax

```
Public Shadows Property Command As SACommand
```

### C# syntax

```
public new SACommand Command {get;set;}
```

## SATcpOptionsBuilder class

Provides a simple way to create and manage the TCP options portion of connection strings used by the SAConnection object.

### Visual Basic syntax

```
Public NotInheritable Class SATcpOptionsBuilder  
    Inherits SAConnectionStringBuilderBase
```

### C# syntax

```
public sealed class SATcpOptionsBuilder : SAConnectionStringBuilderBase
```

### Base classes

- [“SAConnectionStringBuilderBase class \[SQL Anywhere .NET\]” on page 198](#)

### Members

All members of SATcpOptionsBuilder class, including all inherited members.

Name	Description
<a href="#">SATcpOptionsBuilder constructor</a>	Initializes an SATcpOptionsBuilder object.
<a href="#">Add method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Adds an entry with the specified key and value into the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
<a href="#">AppendKeyValuePair method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Provides an efficient and safe way to append a key and value to an existing <a href="#">System.Text.StringBuilder</a> object.
<a href="#">Clear method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the contents of the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> instance.
<a href="#">ClearPropertyDescriptor method</a> (Inherited from System.Data.Common.DbConnectionStringBuilder)	Clears the collection of <a href="#">System.ComponentModel.PropertyDescriptor</a> objects on the associated <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .

Name	Description
<a href="#">ContainsKey method</a>	Determines whether the <code>SACConnectionStringBuilder</code> object contains a specific keyword.
<a href="#">EquivalentTo method</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Compares the connection information in this <code>System.Data.Common.DbConnectionStringBuilder</code> object with the connection information in the supplied object.
<a href="#">GetKeyword method</a>	Gets the keyword for specified <code>SACConnectionStringBuilder</code> property.
<a href="#">GetProperties method</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Fills a supplied <code>System.Collections.Hashtable</code> with information about all the properties of this <code>System.Data.Common.DbConnectionStringBuilder</code> .
<a href="#">GetUseLongNameAsKeyword method</a>	Gets a boolean values that indicates whether long connection parameter names are used in the connection string.
<a href="#">Remove method</a>	Removes the entry with the specified key from the <code>SACConnectionStringBuilder</code> instance.
<a href="#">SetUseLongNameAsKeyword method</a>	Sets a boolean value that indicates whether long connection parameter names are used in the connection string.
<a href="#">ShouldSerialize method</a>	Indicates whether the specified key exists in this <code>SACConnectionStringBuilder</code> instance.
<a href="#">ToString method</a>	Converts the <code>TcpOptionsBuilder</code> object to a string representation.
<a href="#">TryGetValue method</a>	Retrieves a value corresponding to the supplied key from this <code>SACConnectionStringBuilder</code> .
<a href="#">Broadcast property</a>	Gets or sets the <code>Broadcast</code> option.
<a href="#">BroadcastListener property</a>	Gets or sets the <code>BroadcastListener</code> option.
<a href="#">BrowsableConnectionString property</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Gets or sets a value that indicates whether the <code>System.Data.Common.DbConnectionStringBuilder.ConnectionString</code> property is visible in Visual Studio designers.
<a href="#">ClientPort property</a>	Gets or sets the <code>ClientPort</code> option.
<a href="#">ConnectionString property</a> (Inherited from <code>System.Data.Common.DbConnectionStringBuilder</code> )	Gets or sets the connection string associated with the <code>System.Data.Common.DbConnectionStringBuilder</code> .

Name	Description
Count property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets the current number of keys that are contained within the <a href="#">System.Data.Common.DbConnectionStringBuilder.ConnectionString</a> property.
DoBroadcast property	Gets or sets the DoBroadcast option.
Host property	Gets or sets the Host option.
IPV6 property	Gets or sets the IPV6 option.
IsFixedSize property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> has a fixed size.
IsReadOnly property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets a value that indicates whether the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> is read-only.
Keys property	Gets an <a href="#">System.Collections.ICollection</a> that contains the keys in the <a href="#">SAConnectionStringBuilder</a> .
LDAP property	Gets or sets the LDAP option.
LocalOnly property	Gets or sets the LocalOnly option.
MyIP property	Gets or sets the MyIP option.
ReceiveBufferSize property	Gets or sets the ReceiveBufferSize option.
SendBufferSize property	Gets or sets the Send BufferSize option.
ServerPort property	Gets or sets the ServerPort option.
TDS property	Gets or sets the TDS option.
this property	Gets or sets the value of the connection keyword.
Timeout property	Gets or sets the Timeout option.
Values property (Inherited from System.Data.Common.DbConnectionStringBuilder)	Gets an <a href="#">System.Collections.ICollection</a> that contains the values in the <a href="#">System.Data.Common.DbConnectionStringBuilder</a> .
VerifyServerName property	Gets or sets the VerifyServerName option.

**Remarks**

The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

**See also**

- [“SACConnection class \[SQL Anywhere .NET\]” on page 160](#)

## SATcpOptionsBuilder constructor

Initializes an SATcpOptionsBuilder object.

**Overload list**

Name	Description
<a href="#">SATcpOptionsBuilder() constructor</a>	Initializes an SATcpOptionsBuilder object.
<a href="#">SATcpOptionsBuilder(string) constructor</a>	Initializes an SATcpOptionsBuilder object.

### SATcpOptionsBuilder() constructor

Initializes an SATcpOptionsBuilder object.

**Visual Basic syntax**

```
Public Sub New()
```

**C# syntax**

```
public SATcpOptionsBuilder()
```

**Remarks**

The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

**Example**

The following statement initializes an SATcpOptionsBuilder object.

```
SATcpOptionsBuilder options = new SATcpOptionsBuilder( );
```

### SATcpOptionsBuilder(string) constructor

Initializes an SATcpOptionsBuilder object.

**Visual Basic syntax**

```
Public Sub New(ByVal options As String)
```

**C# syntax**

```
public SATcpOptionsBuilder(string options)
```

### Parameters

- **options** A SQL Anywhere TCP connection parameter options string. For a list of connection parameters, see [“Connection parameters”](#) [*SQL Anywhere Server - Database Administration*].

### Remarks

The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

### Example

The following statement initializes an SATcpOptionsBuilder object.

```
SATcpOptionsBuilder options = new SATcpOptionsBuilder( );
```

## ToString method

Converts the TcpOptionsBuilder object to a string representation.

### Visual Basic syntax

```
Public Overrides Function ToString() As String
```

### C# syntax

```
public override string ToString()
```

### Returns

The options string being built.

## Broadcast property

Gets or sets the Broadcast option.

### Visual Basic syntax

```
Public Property Broadcast As String
```

### C# syntax

```
public string Broadcast {get;set;}
```

## BroadcastListener property

Gets or sets the BroadcastListener option.

### Visual Basic syntax

```
Public Property BroadcastListener As String
```

**C# syntax**

```
public string BroadcastListener {get;set;}
```

## ClientPort property

Gets or sets the ClientPort option.

**Visual Basic syntax**

```
Public Property ClientPort As String
```

**C# syntax**

```
public string ClientPort {get;set;}
```

## DoBroadcast property

Gets or sets the DoBroadcast option.

**Visual Basic syntax**

```
Public Property DoBroadcast As String
```

**C# syntax**

```
public string DoBroadcast {get;set;}
```

## Host property

Gets or sets the Host option.

**Visual Basic syntax**

```
Public Property Host As String
```

**C# syntax**

```
public string Host {get;set;}
```

## IPV6 property

Gets or sets the IPV6 option.

**Visual Basic syntax**

```
Public Property IPV6 As String
```

### **C# syntax**

```
public string IPV6 {get;set;}
```

## **LDAP property**

Gets or sets the LDAP option.

### **Visual Basic syntax**

```
Public Property LDAP As String
```

### **C# syntax**

```
public string LDAP {get;set;}
```

## **LocalOnly property**

Gets or sets the LocalOnly option.

### **Visual Basic syntax**

```
Public Property LocalOnly As String
```

### **C# syntax**

```
public string LocalOnly {get;set;}
```

## **MyIP property**

Gets or sets the MyIP option.

### **Visual Basic syntax**

```
Public Property MyIP As String
```

### **C# syntax**

```
public string MyIP {get;set;}
```

## **ReceiveBufferSize property**

Gets or sets the ReceiveBufferSize option.

### **Visual Basic syntax**

```
Public Property ReceiveBufferSize As Integer
```



**C# syntax**

```
public int ReceiveBufferSize {get;set;}
```

## SendBufferSize property

Gets or sets the Send BufferSize option.

**Visual Basic syntax**

```
Public Property SendBufferSize As Integer
```

**C# syntax**

```
public int SendBufferSize {get;set;}
```

## ServerPort property

Gets or sets the ServerPort option.

**Visual Basic syntax**

```
Public Property ServerPort As String
```

**C# syntax**

```
public string ServerPort {get;set;}
```

## TDS property

Gets or sets the TDS option.

**Visual Basic syntax**

```
Public Property TDS As String
```

**C# syntax**

```
public string TDS {get;set;}
```

## Timeout property

Gets or sets the Timeout option.

**Visual Basic syntax**

```
Public Property Timeout As Integer
```

### C# syntax

```
public int Timeout {get;set;}
```

## VerifyServerName property

Gets or sets the VerifyServerName option.

### Visual Basic syntax

```
Public Property VerifyServerName As String
```

### C# syntax

```
public string VerifyServerName {get;set;}
```

## SATransaction class

Represents a SQL transaction.

### Visual Basic syntax

```
Public NotInheritable Class SATransaction  
    Inherits System.Data.Common.DbTransaction
```

### C# syntax

```
public sealed class SATransaction : System.Data.Common.DbTransaction
```

### Base classes

- [System.Data.Common.DbTransaction](#)

### Members

All members of SATransaction class, including all inherited members.

Name	Description
<a href="#">Commit method</a>	Commits the database transaction.
<a href="#">Dispose method</a> (Inherited from System.Data.Common.DbTransaction)	Releases the unmanaged resources used by the <a href="#">System.Data.Common.DbTransaction</a> .
<a href="#">Rollback method</a>	Rolls back a transaction from a pending state.
<a href="#">Save method</a>	Creates a savepoint in the transaction that can be used to roll back a portion of the transaction, and specifies the savepoint name.

Name	Description
<a href="#">Connection property</a>	The SAConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
<a href="#">IsolationLevel property</a>	Specifies the isolation level for this transaction.
<a href="#">SAIsolationLevel property</a>	Specifies the isolation level for this transaction.

**Remarks**

There is no constructor for SATransaction. To obtain an SATransaction object, use one of the BeginTransaction methods. To associate a command with a transaction, use the SACommand.Transaction property.

For more information, see [“Transaction processing” on page 62](#) and [“Manipulate data using the SACommand object” on page 48](#).

**See also**

- [“SAConnection.BeginTransaction method \[SQL Anywhere .NET\]” on page 163](#)
- [“SACommand.Transaction property \[SQL Anywhere .NET\]” on page 143](#)

## Commit method

Commits the database transaction.

**Visual Basic syntax**

```
Public Overrides Sub Commit()
```

**C# syntax**

```
public override void Commit()
```

## Rollback method

Rolls back a transaction from a pending state.

**Overload list**

Name	Description
<a href="#">Rollback() method</a>	Rolls back a transaction from a pending state.
<a href="#">Rollback(string) method</a>	Rolls back a transaction from a pending state.

## Rollback() method

Rolls back a transaction from a pending state.

### Visual Basic syntax

```
Public Overrides Sub Rollback()
```

### C# syntax

```
public override void Rollback()
```

### Remarks

The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

## Rollback(string) method

Rolls back a transaction from a pending state.

### Visual Basic syntax

```
Public Sub Rollback(ByVal savePoint As String)
```

### C# syntax

```
public void Rollback(string savePoint)
```

### Parameters

- **savePoint** The name of the savepoint to roll back to.

### Remarks

The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

## Save method

Creates a savepoint in the transaction that can be used to roll back a portion of the transaction, and specifies the savepoint name.

### Visual Basic syntax

```
Public Sub save(ByVal savePoint As String)
```

### C# syntax

```
public void save(string savePoint)
```

### Parameters

- **savePoint** The name of the savepoint to which to roll back.

## Connection property

The SAConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

### Visual Basic syntax

```
Public ReadOnly Shadows Property Connection As SAConnection
```

### C# syntax

```
public new SAConnection Connection {get;}
```

### Remarks

A single application can have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction.

## IsolationLevel property

Specifies the isolation level for this transaction.

### Visual Basic syntax

```
Public ReadOnly Overrides Property IsolationLevel As System.Data.IsolationLevel
```

### C# syntax

```
public override System.Data.IsolationLevel IsolationLevel {get;}
```

### Remarks

The isolation level for this transaction. This can be one of:

- ReadCommitted
- ReadUncommitted
- RepeatableRead
- Serializable
- Snapshot
- ReadOnlySnapshot
- StatementSnapshot

The default is ReadCommitted.

## SAIsolationLevel property

Specifies the isolation level for this transaction.

### Visual Basic syntax

```
Public ReadOnly Property SAIsolationLevel As SAIsolationLevel
```

### C# syntax

```
public SAIsolationLevel SAIsolationLevel {get;}
```

### Remarks

The IsolationLevel for this transaction. This can be one of:

- Chaos
- Read ReadCommitted
- ReadOnlySnapshot
- ReadUncommitted
- RepeatableRead
- Serializable
- Snapshot
- StatementSnapshot
- Unspecified

The default is ReadCommitted.

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

## SAInfoMessageEventHandler delegate

Represents the method that handles the SAConnection.InfoMessage event of an SAConnection object.

### Visual Basic syntax

```
Public Delegate Sub SAInfoMessageEventHandler(  
    ByVal obj As Object,  
    ByVal args As SAInfoMessageEventArgs  
)
```

### C# syntax

```
public delegate void SAInfoMessageEventHandler(  
    object obj,
```

```
        SAInfoMessageEventArgs args
    );
```

**See also**

- [“SAConnection class \[SQL Anywhere .NET\]” on page 160](#)
- [“SAConnection.InfoMessage event \[SQL Anywhere .NET\]” on page 180](#)

## SARowUpdatedEventHandler delegate

Represents the method that handles the RowUpdated event of an SADataAdapter.

**Visual Basic syntax**

```
Public Delegate Sub SARowUpdatedEventHandler(  
    ByVal sender As Object,  
    ByVal e As SARowUpdatedEventArgs  
)
```

**C# syntax**

```
public delegate void SARowUpdatedEventHandler(  
    object sender,  
    SARowUpdatedEventArgs e  
);
```

## SARowUpdatingEventHandler delegate

Represents the method that handles the RowUpdating event of an SADataAdapter.

**Visual Basic syntax**

```
Public Delegate Sub SARowUpdatingEventHandler(  
    ByVal sender As Object,  
    ByVal e As SARowUpdatingEventArgs  
)
```

**C# syntax**

```
public delegate void SARowUpdatingEventHandler(  
    object sender,  
    SARowUpdatingEventArgs e  
);
```

## SARowsCopiedEventHandler delegate

Represents the method that handles the SABulkCopy.SARowsCopied event of an SABulkCopy.

### Visual Basic syntax

```
Public Delegate Sub SARowsCopiedEventHandler(  
    ByVal sender As Object,  
    ByVal rowsCopiedEventArgs As SARowsCopiedEventArgs  
)
```

### C# syntax

```
public delegate void SARowsCopiedEventHandler(  
    object sender,  
    SARowsCopiedEventArgs rowsCopiedEventArgs  
) ;
```

### Remarks

The SARowsCopiedEventHandler delegate is not available in the .NET Compact Framework 2.0.

### See also

- [“SABulkCopy class \[SQL Anywhere .NET\]” on page 99](#)

## SABulkCopyOptions enumeration

A bitwise flag that specifies one or more options to use with an instance of SABulkCopy.

### Visual Basic syntax

```
Public Enum SABulkCopyOptions
```

### C# syntax

```
public enum SABulkCopyOptions
```

### Members

Member name	Description	Value
Default	Specifying only this value causes the default behavior to be used.  By default, triggers are enabled.	0x0
DoNotFireTriggers	When specified, triggers are not fired.  Disabling triggers requires DBA permission. Triggers are disabled for the connection at the start of WriteToServer and the value is restored at the end of the method.	0x1
KeepIdentity	When specified, the source values to be copied into an identity column are preserved.  By default, new identity values are generated in the destination table.	0x2



Member name	Description	Value
TableLock	When specified the table is locked using the command LOCK TABLE table_name WITH HOLD IN SHARE MODE.  This lock is in place until the connection is closed.	0x4
UseInternal-Transaction	When specified, each batch of the bulk-copy operation is executed within a transaction.  When not specified, transaction aren't used. If you indicate this option and also provide an SATransaction object to the constructor, a System.ArgumentException occurs.	0x8

### Remarks

The SABulkCopyOptions enumeration is used when you construct an SABulkCopy object to specify how the WriteToServer methods will behave.

The SABulkCopyOptions class is not available in the .NET Compact Framework 2.0.

The CheckConstraints and KeepNulls options are not supported.

### See also

- [“SABulkCopy class \[SQL Anywhere .NET\]” on page 99](#)

## SADbType enumeration

Enumerates the SQL Anywhere .NET database data types.

### Visual Basic syntax

```
Public Enum SADbType
```

### C# syntax

```
public enum SADbType
```

### Members

Member name	Description
BigInt	Signed 64-bit integer.
Binary	Binary data, with a specified maximum length.  The enumeration values Binary and VarBinary are aliases of each other.

<b>Member name</b>	<b>Description</b>
Bit	1-bit flag.
Char	Character data, with a specified length.  This type always supports Unicode characters. The types Char and VarChar are fully compatible.
Date	Date information.
DateTime	Timestamp information (date, time).  The enumeration values DateTime and TimeStamp are aliases of each other.
DateTimeOffset	Timestamp information (date, time) offset.
Decimal	Exact numerical data, with a specified precision and scale.  The enumeration values Decimal and Numeric are aliases of each other.
Double	Double-precision floating-point number (8 bytes).
Float	Single-precision floating-point number (4 bytes).  The enumeration values Float and Real are aliases of each other.
Image	Stores binary data of arbitrary length.
Integer	Unsigned 32-bit integer.
LongBinary	Binary data, with variable length.
LongNVarChar	Character data in the NCHAR character set, with variable length.  This type always supports Unicode characters.
LongVarbit	Bit arrays, with variable length.
LongVarChar	Character data, with variable length.  This type always supports Unicode characters.
Money	Monetary data.
NChar	Stores Unicode character data, up to 32767 characters.
NText	Stores Unicode character data of arbitrary length.

Member name	Description
Numeric	Exact numerical data, with a specified precision and scale. The enumeration values Decimal and Numeric are aliases of each other.
NVarChar	Stores Unicode character data, up to 32767 characters.
Real	Single-precision floating-point number (4 bytes). The enumeration values Float and Real are aliases of each other.
SmallDateTime	A domain, implemented as <code>TIMESTAMP</code> .
SmallInt	Signed 16-bit integer.
SmallMoney	Stores monetary data that is less than one million currency units.
SysName	Stores character data of arbitrary length.
Text	Stores character data of arbitrary length.
Time	Time information.
TimeStamp	Timestamp information (date, time). The enumeration values DateTime and TimeStamp are aliases of each other.
TimeStampWith- TimeZone	Timestamp information (date, time, time zone). The enumeration values DateTime and TimeStamp are aliases of each other.
TinyInt	Unsigned 8-bit integer.
UniqueIdentifier	Universally Unique Identifier (UUID/GUID).
UniqueIdent- fierStr	A domain, implemented as <code>CHAR( 36 )</code> . UniqueIdentifierStr is used for remote data access when mapping Microsoft SQL Server uniqueidentifier columns.
UnsignedBigInt	Unsigned 64-bit integer.
UnsignedInt	Unsigned 32-bit integer.
UnsignedSmallInt	Unsigned 16-bit integer.
VarBinary	Binary data, with a specified maximum length. The enumeration values <b>Binary</b> and <b>VarBinary</b> are aliases of each other.

Member name	Description
VarBit	Bit arrays that are from 1 to 32767 bits in length.
VarChar	Character data, with a specified maximum length.  This type always supports Unicode characters. The types Char and VarChar are fully compatible.
Xml	XML data.  This type stores character data of arbitrary length, and is used to store XML documents.

**Remarks**

The table below lists which .NET types are compatible with each SADBType. In the case of integral types, table columns can always be set using smaller integer types, but can also be set using larger types as long as the actual value is within the range of the type.

SADBType	Compatible .NET type	C# built-in type	Visual Basic built-in type
<b>BigInt</b>	System.Int64	long	Long
<b>Binary, VarBinary</b>	System.Byte[], or System.Guid if size is 16	byte[]	Byte()
<b>Bit</b>	System.Boolean	bool	Boolean
<b>Char, VarChar</b>	System.String	String	String
<b>Date</b>	System.DateTime	DateTime (no built-in type)	Date
<b>DateTime, TimeStamp</b>	System.DateTime	DateTime (no built-in type)	DateTime
<b>DateTimeOffset, Date-TimeOffset</b>	System.DateTimeOffset	DateTimeOffset (no built-in type)	DateTimeOffset
<b>Decimal, Numeric</b>	System.String	decimal	Decimal
<b>Double</b>	System.Double	double	Double
<b>Float, Real</b>	System.Single	float	Single
<b>Image</b>	System.Byte[]	byte[]	Byte()

<b>SADbType</b>	<b>Compatible .NET type</b>	<b>C# built-in type</b>	<b>Visual Basic built-in type</b>
<b>Integer</b>	System.Int32	int	Integer
<b>LongBinary</b>	System.Byte[]	byte[]	Byte()
<b>LongNVarChar</b>	System.String	String	String
<b>LongVarChar</b>	System.String	String	String
<b>Money</b>	System.String	decimal	Decimal
<b>NChar</b>	System.String	String	String
<b>NText</b>	System.String	String	String
<b>Numeric</b>	System.String	decimal	Decimal
<b>NVarChar</b>	System.String	String	String
<b>SmallDateTime</b>	System.DateTime	DateTime (no built-in type)	Date
<b>SmallInt</b>	System.Int16	short	Short
<b>SmallMoney</b>	System.String	decimal	Decimal
<b>SysName</b>	System.String	String	String
<b>Text</b>	System.String	String	String
<b>Time</b>	System.TimeSpan	TimeSpan (no built-in type)	TimeSpan (no built-in type)
<b>TimeStamp</b>	System.DateTime	DateTime (no built-in type)	Date
<b>TimeStampWithTime-Zone</b>	System.DateTimeOffset	DateTimeOffset (no built-in type)	DateTimeOffset
<b>TinyInt</b>	System.Byte	byte	Byte
<b>UniqueIdentifier</b>	System.Guid	Guid (no built-in type)	Guid (no built-in type)
<b>UniqueIdentifierStr</b>	System.String	String	String

SADBType	Compatible .NET type	C# built-in type	Visual Basic built-in type
<b>UnsignedBigInt</b>	System.UInt64	ulong	UInt64 (no built-in type)
<b>UnsignedInt</b>	System.UInt32	uint	UInt64 (no built-in type)
<b>UnsignedSmallInt</b>	System.UInt16	ushort	UInt64 (no built-in type)
<b>Xml</b>	System.Xml	String	String

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

**See also**

- [“SADbDataReader.GetFieldType method \[SQL Anywhere .NET\]” on page 225](#)
- [“SADbDataReader.GetDataTypeName method \[SQL Anywhere .NET\]” on page 222](#)

## SAIsolationLevel enumeration

Specifies SQL Anywhere isolation levels.

**Visual Basic syntax**

```
Public Enum SAIsolationLevel
```

**C# syntax**

```
public enum SAIsolationLevel
```

**Members**

Member name	Description
Chaos	<p>This isolation level is unsupported.</p> <p>If you call <code>SADbConnection.BeginTransaction</code> with this isolation level, an exception is thrown.</p> <p>For more information, see <a href="#">“SADbConnection.BeginTransaction method [SQL Anywhere .NET]” on page 163</a>.</p>

Member name	Description
ReadCommitted	<p>Sets the behavior to be equivalent to isolation level 1.</p> <p>This isolation level prevents dirty reads, but allows non-repeatable reads and phantom rows.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
ReadUncommitted	<p>Sets the behavior to be equivalent to isolation level 0.</p> <p>This isolation level allows dirty reads, non-repeatable reads, and phantom rows.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
Repeatable-Read	<p>Sets the behavior to be equivalent to isolation level 2.</p> <p>This isolation level prevents dirty reads and guarantees repeatable reads. However, it allows phantom rows.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
Serializable	<p>Sets the behavior to be equivalent to isolation level 3.</p> <p>This isolation level prevents dirty reads, guarantees repeatable reads, and prevents phantom rows.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
Snapshot	<p>Uses a snapshot of committed data from the time when the first row is read, inserted, updated, or deleted by the transaction.</p> <p>This isolation level uses a snapshot of committed data from the time when the first row is read or updated by the transaction.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
Unspecified	<p>This isolation level is unsupported.</p> <p>If you call <code>SACConnection.BeginTransaction</code> with this isolation level, an exception is thrown.</p> <p>For more information, see <a href="#">“SACConnection.BeginTransaction method [SQL Anywhere .NET]” on page 163</a>.</p>

Member name	Description
ReadOnlySnapshot	<p>For read-only statements, use a snapshot of committed data from the time when the first row is read from the database.</p> <p>Non-repeatable reads and phantom rows can occur within a transaction, but not within a single statement. For updatable statements, use the isolation level specified by the <code>updatable_statement_isolation</code> option (can be one of 0 (the default), 1, 2, or 3).</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>
StatementSnapshot	<p>Use a snapshot of committed data from the time when the first row is read by the statement.</p> <p>Each statement within the transaction sees a snapshot of data from a different time.</p> <p>For each statement, use a snapshot of committed data from the time when the first row is read from the database. Non-repeatable reads and phantom rows can occur within a transaction, but not within a single statement.</p> <p>For more information, see <a href="#">“Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]</a>.</p>

**Remarks**

This class augments the `System.Data.IsolationLevel` class.

The SQL Anywhere .NET Data Provider supports all SQL Anywhere isolation levels, including the snapshot isolation levels. To use snapshot isolation, specify one of `SAIsolationLevel.Snapshot`, `SAIsolationLevel.ReadOnlySnapshot`, or `SAIsolationLevel.StatementSnapshot` as the parameter to `BeginTransaction`. `BeginTransaction` has been overloaded so it can take either an `IsolationLevel` or an `SAIsolationLevel`. The values in the two enumerations are the same, except for `ReadOnlySnapshot` and `StatementSnapshot` which exist only in `SAIsolationLevel`. There is a new property in `SATransaction` called `SAIsolationLevel` that gets the `SAIsolationLevel`.

For more information, see [“Snapshot isolation” \[SQL Anywhere Server - SQL Usage\]](#).

## SAMessageType enumeration

Identifies the type of message.

**Visual Basic syntax**

```
Public Enum SAMessageType
```

**C# syntax**

```
public enum SAMessageType
```



**Members**

<b>Member name</b>	<b>Description</b>	<b>Value</b>
Action	Message of type ACTION.	2
Info	Message of type INFO.	0
Status	Message of type STATUS.	3
Warning	Message of type WARNING.	1

**Remarks**

This can be one of: Action, Info, Status, or Warning.

---

---

# OLE DB and ADO development

SQL Anywhere includes an OLE DB provider for OLE DB and ADO.

OLE DB is a set of Component Object Model (COM) interfaces developed by Microsoft, which provide applications with uniform access to data stored in diverse information sources and that also provide the ability to implement additional database services. These interfaces support the amount of DBMS functionality appropriate to the data store, enabling it to share its data.

ADO is an object model for programmatically accessing, editing, and updating a wide variety of data sources through OLE DB system interfaces. ADO is also developed by Microsoft. Most developers using the OLE DB programming interface do so by writing to the ADO API rather than directly to the OLE DB API.

Do not confuse the ADO interface with ADO.NET. ADO.NET is a separate interface.

Refer to the Microsoft Developer Network for documentation on OLE DB and ADO programming. For SQL Anywhere-specific information about OLE DB and ADO development, use this document.

## See also

- [“SQL Anywhere .NET support” on page 39](#)

## OLE DB

OLE DB is a data access model from Microsoft. It uses the Component Object Model (COM) interfaces and, unlike ODBC, OLE DB does not assume that the data source uses a SQL query processor.

SQL Anywhere includes an **OLE DB provider** named **SAOLEDB**. This provider is available for current Windows platforms. The provider is not available for Windows Mobile platforms.

You can also access SQL Anywhere using the Microsoft OLE DB Provider for ODBC (MSDASQL), together with the SQL Anywhere ODBC driver.

Using the SQL Anywhere OLE DB provider brings several benefits:

- Some features, such as updating through a cursor, are not available using the OLE DB/ODBC bridge.
- If you use the SQL Anywhere OLE DB provider, ODBC is not required in your deployment.
- MSDASQL allows OLE DB clients to work with any ODBC driver, but does not guarantee that you can use the full range of functionality of each ODBC driver. Using the SQL Anywhere provider, you can get full access to SQL Anywhere features from OLE DB programming environments.

## Supported platforms

The SQL Anywhere OLE DB provider is designed to work with Microsoft Data Access Components (MDAC) 2.8 and later versions.

For a list of supported platforms, see <http://www.sybase.com/detail?id=1061806>.

## Distributed transactions in OLE DB

The OLE DB driver can be used as a resource manager in a distributed transaction environment.

### See also

- “Three-tier computing and distributed transactions” on page 839

## ADO programming with SQL Anywhere

ADO (ActiveX Data Objects) is a data access object model exposed through an Automation interface, which allows client applications to discover the methods and properties of objects at runtime without any prior knowledge of the object. Automation allows scripting languages like Visual Basic to use a standard data access object model. ADO uses OLE DB to provide data access.

Using the SQL Anywhere OLE DB provider, you get full access to SQL Anywhere features from an ADO programming environment.

This section describes how to perform basic tasks while using ADO from Visual Basic. It is not a complete guide to programming using ADO.

Code samples from this section can be found in the *%SQLANYXSAMP12%\SQLAnywhere\VBSampler\vbsampler.sln* project file.

For information about programming in ADO, see your development tool documentation.

## Connecting to a database with the Connection object

This section describes a simple Visual Basic routine that connects to a database.

### Sample code

You can try this routine by placing a command button named cmdTestConnection on a form, and pasting the routine into its Click event. Run the program and click the button to connect and then disconnect.

```
Private Sub cmdTestConnection_Click( _  
    ByVal eventSender As System.Object, _  
    ByVal eventArgs As System.EventArgs) _  
    Handles cmdTestConnection.Click
```

```

' Declare variables
Dim myConn As New ADODB.Connection
Dim myCommand As New ADODB.Command
Dim cAffected As Integer

On Error GoTo HandleError

' Establish the connection
myConn.Provider = "SAOLEDB"
myConn.ConnectionString = _
    "Data Source=SQL Anywhere 12 Demo"
myConn.Open()
MsgBox("Connection succeeded")
myConn.Close()
Exit Sub

HandleError:
MsgBox(ErrorToString(Err.Number))
Exit Sub
End Sub

```

**Notes**

The sample carries out the following tasks:

- It declares the variables used in the routine.
- It establishes a connection, using the SQL Anywhere OLE DB provider, to the sample database.
- It uses a Command object to execute a simple statement, which displays a message in the database server messages window.
- It closes the connection.

**See also**

- [“OLE DB connection parameters” on page 336](#)

## Executing statements with the Command object

This section describes a simple routine that sends a simple SQL statement to the database.

**Sample code**

You can try this routine by placing a command button named cmdUpdate on a form, and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, and then disconnect.

```

Private Sub cmdUpdate_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdUpdate.Click

' Declare variables
Dim myConn As New ADODB.Connection
Dim myCommand As New ADODB.Command
Dim cAffected As Integer

```

```
On Error GoTo HandleError

' Establish the connection
myConn.Provider = "SAOLEDB"
myConn.ConnectionString = _
    "Data Source=SQL Anywhere 12 Demo"
myConn.Open()

'Execute a command
myCommand.CommandText = _
    "UPDATE Customers SET GivenName='Liz' WHERE ID=102"
myCommand.ActiveConnection = myConn
myCommand.Execute(cAffected)
MsgBox(CStr(cAffected) & " rows affected.", _
    MsgBoxStyle.Information)

myConn.Close()
Exit Sub

HandleError:
MsgBox(ErrorToString(Err.Number))
Exit Sub
End Sub
```

## Notes

After establishing a connection, the example code creates a Command object, sets its CommandText property to an update statement, and sets its ActiveConnection property to the current connection. It then executes the update statement and displays the number of rows affected by the update in a window.

In this example, the update is sent to the database and committed when it is executed.

You can also perform updates through a cursor.

## See also

- [“ADO transactions” on page 335](#)
- [“Updating data through a cursor” on page 334](#)

## Querying the database with the Recordset object

The ADO Recordset object represents the result set of a query. You can use it to view data from a database.

## Sample code

You can try this routine by placing a command button named cmdQuery on a form and pasting the routine into its Click event. Run the program and click the button to connect, display a message in the database server messages window, execute a query and display the first few rows in windows, and then disconnect.

```
Private Sub cmdQuery_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdQuery.Click

    ' Declare variables
```

```

Dim i As Integer
Dim myConn As New ADODB.Connection
Dim myCommand As New ADODB.Command
Dim myRS As New ADODB.Recordset

On Error GoTo ErrorHandler

' Establish the connection
myConn.Provider = "SAOLEDB"
myConn.ConnectionString = _
    "Data Source=SQL Anywhere 12 Demo"
myConn.CursorLocation = _
    ADODB.CursorLocationEnum.adUseServer
myConn.Mode = _
    ADODB.ConnectModeEnum.adModeReadWrite
myConn.IsolationLevel = _
    ADODB.IsolationLevelEnum.adXactCursorStability
myConn.Open()

'Execute a query
myRS = New ADODB.Recordset
myRS.CacheSize = 50
myRS.Provider = "SAOLEDB"
myRS.ActiveConnection = myConn
myRS.CursorType = ADODB.CursorTypeEnum.adOpenKeyset
myRS.LockType = ADODB.LockTypeEnum.adLockOptimistic
myRS.Open()

' Scroll through the first few results
myRS.MoveFirst()
For i = 1 To 5
    MsgBox(myRS.Fields("CompanyName").Value, _
        MsgBoxStyle.Information)
    myRS.MoveNext()
Next

myRS.Close()
myConn.Close()
Exit Sub

ErrorHandler:
MsgBox(ErrorToString(Err.Number))
Exit Sub
End Sub

```

**Notes**

The Recordset object in this example holds the results from a query on the Customers table. The For loop scrolls through the first several rows and displays the CompanyName value for each row.

This is a simple example of using a cursor from ADO.

For more advanced examples of using a cursor from ADO, see [“The Recordset object” on page 333](#).

## The Recordset object

When working with SQL Anywhere, the ADO Recordset represents a cursor. You can choose the type of cursor by declaring a CursorType property of the Recordset object before you open the Recordset. The

choice of cursor type controls the actions you can take on the Recordset and has performance implications.

### Cursor types

ADO has its own naming convention for cursor types. The set of cursor types supported by SQL Anywhere is described in [“Cursor properties” on page 15](#).

The available cursor types, the corresponding cursor type constants, and the SQL Anywhere types they are equivalent to, are as follows:

ADO cursor type	ADO constant	SQL Anywhere type
Dynamic cursor	adOpenDynamic	Dynamic scroll cursor
Keyset cursor	adOpenKeyset	Scroll cursor
Static cursor	adOpenStatic	Insensitive cursor
Forward only	adOpenForwardOnly	No-scroll cursor

### Sample code

The following code sets the cursor type for an ADO Recordset object:

```
Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

### See also

- [“Cursor types” on page 14](#)

## Updating data through a cursor

The SQL Anywhere OLE DB provider lets you update a result set through a cursor. This capability is not available through the MSDASQL provider.

### Updating record sets

You can update the database through a Recordset.

```
Private Sub cmdUpdateThroughCursor_Click( _
    ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) _
    Handles cmdUpdateThroughCursor.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim SQLString As String

    On Error GoTo HandleError
```



```

' Connect
myConn.Provider = "SAOLEDB"
myConn.ConnectionString = _
    "Data Source=SQL Anywhere 12 Demo"
myConn.Open()
myConn.BeginTrans()
SQLString = "SELECT * FROM Customers"
myRS.Open(SQLString, myConn, _
    ADOB.CursorTypeEnum.adOpenDynamic, _
    ADOB.LockTypeEnum.adLockBatchOptimistic)

If myRS.BOF And myRS.EOF Then
    MsgBox("Recordset is empty!", 16, "Empty Recordset")
Else
    MsgBox("Cursor type: " & CStr(myRS.CursorType), _
        MsgBoxStyle.Information)
    myRS.MoveFirst()
    For i = 1 To 3
        MsgBox("Row: " & CStr(myRS.Fields("ID").Value), _
            MsgBoxStyle.Information)
        If i = 2 Then
            myRS.Update("City", "Toronto")
            myRS.UpdateBatch()
        End If
        myRS.MoveNext()
    Next i
    myRS.Close()
End If
myConn.CommitTrans()
myConn.Close()
Exit Sub

HandleError:
MsgBox(ErrorToString(Err.Number))
Exit Sub

End Sub

```

## Notes

If you use the `adLockBatchOptimistic` setting on the Recordset, the `myRS.Update` method does not make any changes to the database itself. Instead, it updates a local copy of the Recordset.

The `myRS.UpdateBatch` method makes the update to the database server, but does not commit it, because it is inside a transaction. If an `UpdateBatch` method was invoked outside a transaction, the change would be committed.

The `myConn.CommitTrans` method commits the changes. The Recordset object has been closed by this time, so there is no issue of whether the local copy of the data is changed or not.

## ADO transactions

By default, any change you make to the database using ADO is committed when it is executed. This includes explicit updates, and the `UpdateBatch` method on a Recordset. However, the previous section illustrated that you can use the `BeginTrans` and `RollbackTrans` or `CommitTrans` methods on the Connection object to use transactions.

The transaction isolation level is set as a property of the Connection object. The IsolationLevel property can take on one of the following values:

ADO isolation level	Constant	SQL Anywhere level
Unspecified	adXactUnspecified	Not applicable. Set to 0
Chaos	adXactChaos	Unsupported. Set to 0
Browse	adXactBrowse	0
Read uncommitted	adXactReadUncommitted	0
Cursor stability	adXactCursorStability	1
Read committed	adXactReadCommitted	1
Repeatable read	adXactRepeatableRead	2
Isolated	adXactIsolated	3
Serializable	adXactSerializable	3
Snapshot	2097152	4
Statement snapshot	4194304	5
Readonly statement snapshot	8388608	6

**See also**

- [“Isolation levels and consistency” \[SQL Anywhere Server - SQL Usage\]](#)

## OLE DB connection parameters

OLE DB connection parameters are defined by Microsoft. The SQL Anywhere OLE DB provider supports a subset of these connection parameters. A typical connection string looks like this:

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;  
User ID=myUid;Password=myPwd"
```

Below are the OLE DB connection parameters that are supported by the provider. In some cases, OLE DB connection parameters are identical to (for example, Password) or resemble (for example, User ID) SQL Anywhere connection parameters. Note the use of spaces in many of these connection parameters.

- **Provider** This parameter is used to identify the SQL Anywhere OLE DB provider (SAOLEDB).
- **User ID** This connection parameter maps directly to the SQL Anywhere UserID (UID) connection parameter. For example: `User ID=DBA`.

- **Password** This connection parameter maps directly to the SQL Anywhere Password (PWD) connection parameter. For example: `Password=sql`.
- **Data Source** This connection parameter maps directly to the SQL Anywhere DataSourceName (DSN) connection parameter. For example: `Data Source=SQL Anywhere 12 Demo`.
- **Initial Catalog** This connection parameter maps directly to the SQL Anywhere DatabaseName (DBN) connection parameter. For example: `Initial Catalog=demo`.
- **Location** This connection parameter maps directly to the SQL Anywhere Host connection parameter. The parameter value has the same form as the Host parameter value. For example: `Location=localhost:4444`.
- **Extended Properties** This connection parameter is used by OLE DB to pass in all the SQL Anywhere specific connection parameters. For example: `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db"`.

ADO uses this connection parameter to collect and pass in all the connection parameters that it does not recognize.

Some Microsoft connection windows have a field called **Prov String** or **Provider String**. The contents of this field are passed as the value to Extended Properties.

- **OLE DB Services** This connection parameter is not directly handled by the SQL Anywhere OLE DB provider. It controls connection pooling in ADO.
- **Prompt** This connection parameter governs how a connection attempt handles errors. The possible prompt values are 1, 2, 3, or 4. The meanings are DBPROMPT\_PROMPT (1), DBPROMPT\_COMPLETE (2), DBPROMPT\_COMPLETEREQUIRED (3), and DBPROMPT\_NOPROMPT (4).

The default prompt value is 4 which means the provider does not present a connect window. Setting the prompt value to 1 causes a connect window to always appear. Setting the prompt value to 2 causes a connect window to appear if the initial connection attempt fails. Setting the prompt value to 3 causes a connect window to appear if the initial connection attempt fails but the provider disables the controls for any information not required to connect to the data source.

- **Window Handle** The application can pass the handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or the provider does not present any windows. The window handle value is typically 0 (NULL).

Other OLE DB connection parameters can be specified but they are ignored by the OLE DB provider.

When the SQL Anywhere OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is a typical set of property values obtained from Microsoft's RowsetViewer application.

```
User ID 'DBA'  
Password 'sql'  
Location 'localhost:4444'  
Initial Catalog 'demo'
```

```
Data Source 'testds'  
Extended Properties 'appinfo=api=oledb'  
Prompt 2  
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'DSN=testds;HOST=localhost:4444;DBN=demo;UID=DBA;PWD=sql;appinfo=api=oledb'
```

The SQL Anywhere OLE DB provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

This is a simple ADO connection string example.

```
connection.Open "Provider=SAOLEDB;UserID=DBA;Location=localhost:4444;Pwd=sql"
```

ADO parses the connection string and passes all of the unrecognized connection parameters in Extended Properties. When the SQL Anywhere OLE DB provider is invoked, it gets the property values for the OLE DB connection parameters. Here is the set of property values obtained from the ADO application that used the connection string shown above.

```
User ID ''  
Password ''  
Location 'localhost:4444'  
Initial Catalog ''  
Data Source ''  
Extended Properties 'UserID=DBA;Pwd=sql'  
Prompt 4  
Window Handle 0
```

The connection string that the provider constructs from this set of parameter values is:

```
'HOST=localhost:4444;UserID=DBA;Pwd=sql'
```

The provider uses the connection string, Window Handle, and Prompt values as parameters to the database server connection call that it makes.

### See also

- “Userid (UID) connection parameter” [*SQL Anywhere Server - Database Administration*]
- “Password (PWD) connection parameter” [*SQL Anywhere Server - Database Administration*]
- “DataSourceName (DSN) connection parameter” [*SQL Anywhere Server - Database Administration*]
- “DatabaseName (DBN) connection parameter” [*SQL Anywhere Server - Database Administration*]
- “Host connection parameter” [*SQL Anywhere Server - Database Administration*]
- “OLE DB connection pooling” on page 338

## OLE DB connection pooling

The .NET Framework Data Provider for OLE DB automatically pools connections using OLE DB session pooling. When the application closes the connection, it is not actually closed. Instead, the connection is held for a period of time. When your application re-opens a connection, ADO/OLE DB recognizes that the application is using an identical connection string and reuses the open connection. For example, if the

application does an Open/Execute/Close 100 times, there is only 1 actual open and 1 actual close. The final close occurs after about 1 minute of idle time.

If a connection is terminated by external means (such as a forced disconnect using an administrative tool such as Sybase Central), ADO/OLE DB does not know that this has occurred until the next interaction with the server. Caution should be exercised before resorting to forcible disconnects.

The flag that controls connection pooling is `DBPROPVAL_OS_RESOURCEPOOLING` (1). This flag can be turned off using a connection parameter in the connection string.

If you specify **OLE DB Services=-2** in your connection string, then connection pooling is disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

If you specify **OLE DB Services=-4** in your connection string, then connection pooling and transaction enlistment are disabled. Here is a sample connection string:

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

If you disable connection pooling, there is a performance penalty if your application frequently opens and closes connections using the same connection string.

#### See also

- [SQL Server Connection Pooling \(ADO.NET\)](#)
- [Overriding Provider Service Defaults](#)
- [OLE DB, ODBC, and Oracle Connection Pooling \(ADO.NET\)](#)

## Configuration of a Microsoft Linked Server using OLE DB

A Microsoft Linked Server can be created that uses the SQL Anywhere OLE DB provider to obtain access to a SQL Anywhere database. SQL queries can be issued using either the Microsoft four-part table referencing syntax or the Microsoft OPENQUERY SQL function. An example of the four-part syntax follows.

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

In this example, **SADATABASE** is the name of the Linked Server, **demo** is the catalog or database name, **GROUPO** is the table owner in the SQL Anywhere database, and **Customers** is the table name in the SQL Anywhere database.

The other form uses the Microsoft OPENQUERY function.

```
SELECT * FROM OPENQUERY( SADATABASE, 'SELECT * FROM Customers' )
```

In the OPENQUERY syntax, the second SELECT statement ( 'SELECT \* FROM Customers' ) is passed to the SQL Anywhere server for execution.

You can set up a Linked Server that uses the SQL Anywhere OLE DB provider using a Microsoft SQL Server interactive application or a SQL Server script.

**Note**

Before setting up a Linked Server, there are a few things to consider when using Windows Vista or later versions of Windows. SQL Server runs as a service on your system. Depending on how the service is set up on Windows Vista or later versions, a service may not be able to use shared memory connections, it may not be able to start a server, and it may not be able to access User Data Source definitions. For example, a service logged in as a **Network Service** cannot start servers, connect via shared memory, or access User Data Sources. For these situations, the SQL Anywhere server must be started ahead of time and the TCPIP communication protocol must be used. Also, if a data source is to be used, it must be a System Data Source.

### Set up a Linked Server interactively

1. For Microsoft SQL Server 2005/2008, start SQL Server Management Studio. For other versions of SQL Server, the name of this application and the steps to setting up a Linked Server may vary.

In the **Object Explorer** pane, expand **Server Objects » Linked Servers**. Right-click **Linked Servers** and then click **New Linked Server**.

2. Fill in the **General** page.

The **Linked Server** field on the **General** page should contain a **Linked Server** name (like SADATABASE in the example above).

The **Other Data Source** option should be chosen, and **SQL Anywhere OLE DB Provider 12** should be chosen from the **Provider** list.

The **Product Name** field can be anything you like (for example, SQL Anywhere or your application name).

The **Data Source** field can contain an ODBC data source name (DSN). This is a convenience option and a data source name is not required. If you use a System DSN, it must be a 32-bit DSN for 32-bit versions of SQL Server or a 64-bit DSN for 64-bit versions of SQL Server.

`Data Source: SQL Anywhere 12 Demo`

The **Provider String** field can contain additional connection parameters such as UserID (UID), ServerName (Server), and DatabaseFile (DBF).

`Provider string: Server=myserver;DBF=sample.db`

The **Location** field can contain the equivalent of the SQL Anywhere Host connection parameter (for example, localhost:4444 or 10.25.99.253:2638).

`Location: AppServer-pc:2639`

The **Initial Catalog** field can contain the name of the database to connect to (for example, demo). The database must have been previously started.

Initial Catalog: demo

The combination of these last four fields and the user ID and password from the **Security** page must contain enough information to successfully connect to a database server.

3. Instead of specifying the database user ID and password as a connection parameter in the **Provider String** field where it would be exposed in plain text, you can fill in the **Security** page.

In SQL Server 2005/2008, click the **Be made using this security context** option and fill in the **Remote login** and **With password** fields (the password is displayed as asterisks).

4. Go to the **Server Options** page.

Enable the **RPC** and **RPC Out** options.

The technique for doing this varies with different versions of Microsoft SQL Server. In SQL Server 2000, there are two checkboxes that must be checked for these two options. In SQL Server 2005/2008, the options are True/False settings. Make sure that they are set True. The **Remote Procedure Call (RPC)** options must be set if you want to execute stored procedure/function calls in a SQL Anywhere database and pass parameters in and out successfully.

5. Choose the **Allow Inprocess** provider option.

The technique for doing this varies with different versions of Microsoft SQL Server. In SQL Server 2000, there is a **Provider Options** button that takes you to the page where you can choose this option. For SQL Server 2005/2008, right-click the SAOLEDB.12 provider name under **Linked Servers** » **Providers** and click **Properties**. Make sure the **Allow Inprocess** checkbox is checked. If the **Inprocess** option is not chosen, queries fail.

### Set up a Linked Server using a script

- A Linked Server may be set up using a SQL Server script similar to the following. Make the appropriate changes to the script before running it on SQL Server. You should choose a new Linked Server name (SADATABASE is used in the example), an optional data source name (SQL Anywhere 12 Demo is used in the example), an optional provider string (N'host=localhost:4444;server=myserver;dbn=demo' is used in the example), and a remote user ID and password.

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server=N'SADATABASE',
    @srvproduct=N'SQL Anywhere', @provider=N'SAOLEDB.12',
    @datasrc=N'SQL Anywhere 12 Demo',
    @provstr=N'host=localhost:4444;server=myserver;dbn=demo'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc', @optvalue=N'true'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
    @optname=N'rpc out', @optvalue=N'true'
GO
-- Set remote login
EXEC master.dbo.sp_addlinkedsrvlogin @rmtsrvname = N'SADATABASE',
    @locallogin = NULL , @useself = N'False',
```

```

    @rmtuser = N'DBA', @rmtpassword = N'sql'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.12', N'AllowInProcess', 1

```

## Supported OLE DB interfaces

The OLE DB API consists of a set of interfaces. The following table describes the support for each interface in the SQL Anywhere OLE DB driver.

Interface	Purpose	Limitations
IAccessor	Define bindings between client memory and data store values.	DBACCESSOR_PASSBYREF not supported.DBACCESSOR_OPTIMIZED not supported.
IAlterIndex IAlterTable	Alter tables, indexes, and columns.	Not supported.
IChapteredRowset	A chaptered rowset allows rows of a rowset to be accessed in separate chapters.	Not supported. SQL Anywhere does not support chaptered rowsets.
IColumnsInfo	Get simple information about the columns in a rowset.	Supported.
IColumnsRowset	Get information about optional metadata columns in a rowset, and get a rowset of column metadata.	Supported.
ICommand	Execute SQL statements.	Does not support calling. ICommandProperties: GetProperties with DBPROPSET_PROPERTYESINEROR to find properties that could not have been set.
ICommandPersist	Persist the state of a command object (but not any active rowsets). These persistent command objects can subsequently be enumerated using the PROCEDURES or VIEWS rowset.	Supported.
ICommandPrepare	Prepare commands.	Supported.



Interface	Purpose	Limitations
ICommandProperties	Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.	Supported.
ICommandText	Set the SQL statement text for ICommand.	Only the DBGUID_DEFAULT SQL dialect is supported.
ICommandWithParameters	Set or get parameter information for a command.	No support for parameters stored as vectors of scalar values.  No support for BLOB parameters.
IConvertType		Supported.
IDBAsynchNotify IDBAsynchStatus	Asynchronous processing.  Notify client of events in the asynchronous processing of data source initialization, populating rowsets, and so on.	Not supported.
IDBCreateCommand	Create commands from a session.	Supported.
IDBCreateSession	Create a session from a data source object.	Supported.
IDBDataSourceAdmin	Create/destroy/modify data source objects, which are COM objects used by clients. This interface is not used to manage data stores (databases).	Not supported.
IDBInfo	Find information about keywords unique to this provider (that is, to find non-standard SQL keywords).  Also, find information about literals, special characters used in text matching queries, and other literal information.	Supported.
IDBInitialize	Initialize data source objects and enumerators.	Supported.

<b>Interface</b>	<b>Purpose</b>	<b>Limitations</b>
IDBProperties	Manage properties on a data source object or enumerator.	Supported.
IDBSchemaRowset	Get information about system tables, in a standard form (a rowset).	Supported.
IErrorInfo IErrorLookup IErrorRecords	ActiveX error object support.	Supported.
IGetDataSource	Returns an interface pointer to the session's data source object.	Supported.
IIndexDefinition	Create or drop indexes in the data store.	Not supported.
IMultipleResults	Retrieve multiple results (rowsets or row counts) from a command.	Supported.
IOpenRowset	Non-SQL way to access a database table by its name.	Supported.  Opening a table by its name is supported, not by a GUID.
IParentRowset	Access chaptered/hierarchical rowsets.	Not supported.
IRowset	Access rowsets.	Supported.
IRowsetChange	Allow changes to rowset data, reflected back to the data store.  InsertRow/SetData for BLOBs are not implemented.	Supported.
IRowsetChapterMember	Access chaptered/hierarchical rowsets.	Not supported.
IRowsetCurrentIndex	Dynamically change the index for a rowset.	Not supported.
IRowsetFind	Find a row within a rowset matching a specified value.	Not supported.
IRowsetIdentity	Compare row handles.	Not supported.

<b>Interface</b>	<b>Purpose</b>	<b>Limitations</b>
IRowsetIndex	Access database indexes.	Not supported.
IRowsetInfo	Find information about rowset properties or to find the object that created the rowset.	Supported.
IRowsetLocate	Position on rows of a rowset, using bookmarks.	Supported.
IRowsetNotify	Provides a COM callback interface for rowset events.	Supported.
IRowsetRefresh	Get the latest value of data that is visible to a transaction.	Not supported.
IRowsetResynch	Old OLE DB 1.x interface, superseded by IRowsetRefresh.	Not supported.
IRowsetScroll	Scroll through rowset to fetch row data.	Not supported.
IRowsetUpdate	Delay changes to rowset data until Update is called.	Supported.
IRowsetView	Use views on an existing rowset.	Not supported.
ISequentialStream	Retrieve a BLOB column.	Supported for reading only. No support for SetData with this interface.
ISessionProperties	Get session property information.	Supported.
ISourcesRowset	Get a rowset of data source objects and enumerators.	Supported.
ISQLErrorInfo	ActiveX error object support.	Supported.
ISupportErrorInfo		
ITableDefinition ITableDefinitionWithConstraints	Create, drop, and alter tables, with constraints.	Supported.
ITransaction	Commit or abort transactions.	Not all the flags are supported.

Interface	Purpose	Limitations
ITransactionJoin	Support distributed transactions.	Not all the flags are supported.
ITransactionLocal	Handle transactions on a session. Not all the flags are supported.	Supported.
ITransactionOptions	Get or set options on a transaction.	Supported.
IViewChapter	Work with views on an existing rowset, specifically to apply post-processing filters/sorting on rows.	Not supported.
IViewFilter	Restrict contents of a rowset to rows matching a set of conditions.	Not supported.
IViewRowset	Restrict contents of a rowset to rows matching a set of conditions, when opening a rowset.	Not supported.
IViewSort	Apply sort order to a view.	Not supported.

## Registering the OLE DB provider

When the SAOLEDB provider is installed using the SQL Anywhere installer, the provider registers itself. This registration process includes making registry entries in the COM section of the registry, so that ADO can locate the DLL when the SAOLEDB provider is called. If you change the location of your DLL, you must re-register it.

### Register the OLE DB provider

1. Open a command prompt.
2. Change to the directory where the OLE DB provider is installed.
3. Enter the following commands to register the provider:

```
regsvr32 dboledb12.dll  
regsvr32 dboledba12.dll
```

---

## ODBC support

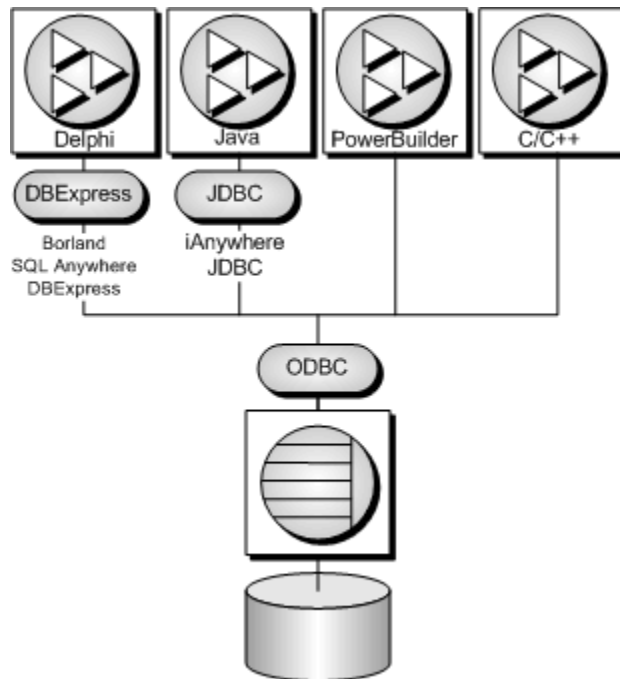
ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft Corporation. It is based on the SQL Access Group CLI specification. ODBC applications can run against any data source that provides an ODBC driver. ODBC is a good choice for a programming interface if you want your application to be portable to other data sources that have ODBC drivers.

ODBC is a low-level interface. Almost all the SQL Anywhere functionality is available with this interface. ODBC is available as a DLL under Windows operating systems. It is provided as a shared object library for Unix.

The core documentation for the ODBC API can be found on the Microsoft Developer Network website at <http://msdn.microsoft.com/en-us/library/ms710252.aspx>.

## Requirements for developing ODBC applications

You can develop applications using a variety of development tools and programming languages, as shown in the figure below, and access the SQL Anywhere database server using the ODBC API.



To write ODBC applications for SQL Anywhere, you need:

- SQL Anywhere.
- A C/C++ compiler capable of creating programs for your environment.

- The Microsoft ODBC Software Development Kit. This is available on the Microsoft Developer Network, and provides documentation and additional tools for testing ODBC applications.

**Note**

Some application development tools, that already have ODBC support, provide their own programming interface that hides the ODBC interface. The SQL Anywhere documentation does not describe how to use those tools.

### The ODBC driver manager

Microsoft Windows includes an ODBC driver manager. There is no driver manager for Windows Mobile. For Unix, a driver manager is supplied with SQL Anywhere.

## ODBC conformance

SQL Anywhere provides support for ODBC 3.5, which is supplied as part of the Microsoft Data Access Kit 2.7.

### Levels of ODBC support

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* at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

### Features supported by SQL Anywhere

SQL Anywhere supports the ODBC 3.5 specification as follows:

- **Core conformance** SQL Anywhere supports all Core level features.
- **Level 1 conformance** SQL Anywhere supports all Level 1 features, except for asynchronous execution of ODBC functions.

SQL Anywhere supports multiple threads sharing a single connection. The requests from the different threads are serialized by SQL Anywhere.

- **Level 2 conformance** SQL Anywhere supports all Level 2 features, except for the following ones:
  - Three part names of tables and views. This is not applicable for SQL Anywhere.
  - Asynchronous execution of ODBC functions for specified individual statements.
  - Ability to time out login requests and SQL queries.

## Building ODBC applications

This section describes how to compile and link simple ODBC applications.

## Including the ODBC header file

Every C/C++ source file that calls ODBC functions must include a platform-specific ODBC header file. Each platform-specific header file includes the main ODBC header file *odbc.h*, which defines all the functions, data types, and constant definitions required to write an ODBC program.

### Include the ODBC header file in a C/C++ source file

1. Add an include line referencing the appropriate platform-specific header file to your source file. The lines to use are as follows:

Operating system	Include line
Windows	#include "ntodbc.h"
Unix	#include "unixodbc.h"
Windows Mobile	#include "ntodbc.h"

2. Add the directory containing the header file to the include path for your compiler.

Both the platform-specific header files and *odbc.h* are installed in the *SDK\Include* subdirectory of your SQL Anywhere installation directory.

3. When building ODBC applications for Unix, you might have to define the macro "UNIX" for 32-bit applications or "UNIX64" for 64-bit applications to obtain the correct data alignment and sizes. This step is not required if you are using one of the following supported compilers:
  - GNU C/C++ compiler on any supported platform
  - Intel C/C++ compiler for Linux (icc)
  - SunPro C/C++ compiler for Linux or Solaris
  - VisualAge C/C++ compiler for AIX
  - C/C++ compiler (cc/aCC) for HP-UX

## Linking ODBC applications on Windows

This section does not apply to Windows Mobile.

When linking your application, you must link against the appropriate import library file to have access to the ODBC functions. The import library defines entry points for the ODBC driver manager *odbc32.dll*. The driver manager in turn loads the SQL Anywhere ODBC driver *dbodbc12.dll*.

### Link an ODBC application (Windows)

1. Add the directory containing the platform-specific import library to the list of library directories in your LIB environment variable.

Typically, the import library is stored under the *Lib* directory structure of the Microsoft platform SDK:

Operating system	Import library
Windows (32-bit)	<i>Lib\odbc32.lib</i>
Windows (64-bit)	<i>Lib\x64\odbc32.lib</i>

Here is an example for a command prompt.

```
set LIB=%LIB%;c:\ms-sdk\v7.0\lib
```

2. To compile and link a simple ODBC application using the Microsoft compile and link tool, you can issue a single command at a command prompt. The following example compiles and links the application stored in *odbc.c*.

```
cl odbc.c /Ic:\sa12\SDK\Include odbc32.lib
```

### See also

- “Linking ODBC applications on Windows Mobile” on page 350

## Linking ODBC applications on Windows Mobile

On Windows Mobile operating systems, there is no ODBC driver manager. The import library (*dbodbc12.lib*) defines entry points directly into the SQL Anywhere ODBC driver *dbodbc12.dll*. This file is located in the *SDK\Lib\CE\Arm.50* subdirectory of your SQL Anywhere installation.

Since there is no ODBC driver manager for Windows Mobile, you must specify the location of the SQL Anywhere ODBC driver DLL with the `DRIVER=` parameter in the connection string supplied to the `SQLDriverConnect` function. The following is an example.

```
szConnStrIn = "driver=\\Windows\\dbodbc12.dll;dbf=\\My Documents\\demo.db"
```

### Link an ODBC application (Windows Mobile)

- Add the directory containing the platform-specific import library to the list of library directories.

For a list of supported versions of Windows Mobile, see the SQL Anywhere for PC Platforms table at <http://www.sybase.com/detail?id=1002288>.

The sample program (*odbc\_sample.cpp*) uses a File Data Source (FileDSN connection parameter) called *SQL Anywhere 12 Demo.dsn*. This file placed in the root directory of your Windows Mobile device when you install SQL Anywhere for Windows Mobile to your device. You can create file data sources on your desktop system with the ODBC Data Source Administrator, but they must be set up for your desktop environment and then edited to match the Windows Mobile environment. After appropriate edits, you can copy them to your Windows Mobile device.



## Windows Mobile and Unicode

SQL Anywhere uses an encoding known as UTF-8, a multibyte character encoding that can be used to encode Unicode.

The SQL Anywhere ODBC driver supports either ASCII (8-bit) strings or Unicode code (wide character) strings. The `UNICODE` macro controls whether ODBC functions expect ASCII or Unicode strings. If your application must be built with the `UNICODE` macro defined, but you want to use the ASCII ODBC functions, then the `SQL_NOUNICODEMAP` macro must also be defined.

The sample file `%SQLANYWHERE%\SQLAnywhere\C\odbc.c` illustrates how to use the Unicode ODBC features.

## Linking ODBC applications on Unix

An ODBC driver manager for Unix is included with SQL Anywhere and there are third party driver managers available. This section describes how to build ODBC applications that do not use an ODBC driver manager.

### ODBC driver

The ODBC driver is a shared object or shared library. Separate versions of the SQL Anywhere ODBC driver are supplied for single-threaded and multithreaded applications. A generic SQL Anywhere ODBC driver is supplied that will detect the threading model in use and direct calls to the appropriate single-threaded or multithreaded library.

The ODBC drivers are the following files:

Operating system	Threading model	ODBC driver
(all Unix except Mac OS X and HP-UX)	Generic	<i>libdbodbc12.so (libdbodbc12.so.1)</i>
(all Unix except Mac OS X and HP-UX)	Single threaded	<i>libdbodbc12_n.so (libdbodbc12_n.so.1)</i>
(all Unix except Mac OS X and HP-UX)	Multithreaded	<i>libdbodbc12_r.so (libdbodbc12_r.so.1)</i>
HP-UX	Generic	<i>libdbodbc12.sl (libdbodbc12.sl.1)</i>
HP-UX	Single threaded	<i>libdbodbc12_n.sl (libdbodbc12_n.sl.1)</i>
HP-UX	Multithreaded	<i>libdbodbc12_r.sl (libdbodbc12_r.sl.1)</i>
Mac OS X	Generic	<i>libdbodbc12.dylib</i>
Mac OS X	Single threaded	<i>libdbodbc12_n.dylib</i>

Operating system	Threading model	ODBC driver
Mac OS X	Multithreaded	<i>libdbodbc12_r.dylib</i>

The libraries are installed as symbolic links to the shared library with a version number (shown in parentheses).

In addition, the following bundles are also provided for Mac OS X:

Operating system	Threading model	ODBC driver
Mac OS X	Single threaded	<i>dbodbc12.bundle</i>
Mac OS X	Multithreaded	<i>dbodbc12_r.bundle</i>

### Link an ODBC application (Unix)

1. Link your application against the generic ODBC driver *libdbodbc12*.
2. When deploying your application, ensure that the appropriate (or all) ODBC driver versions (non-threaded or threaded) are available in the user's library path.

### Data source information

If SQL Anywhere does not detect the presence of an ODBC driver manager, it uses the system information file for data source information.

### see also

- [“ODBC data sources on Unix” \[SQL Anywhere Server - Database Administration\]](#)

## The SQL Anywhere ODBC driver manager for Unix

SQL Anywhere includes an ODBC driver manager for Unix. The *libdbodm12* shared object can be used on all supported Unix platforms as an ODBC driver manager. The iAnywhere ODBC driver manager can be used to load any version 3.0 or later ODBC driver. The driver manager will not perform mappings between ODBC 1.0/2.0 calls and ODBC 3.x calls; therefore, applications using the iAnywhere ODBC driver manager must restrict their use of the ODBC feature set to version 3.0 and later. Also, the iAnywhere ODBC driver manager can be used by both threaded and non-threaded applications.

The iAnywhere ODBC driver manager can perform tracing of ODBC calls for any given connection. To turn on tracing, a user can use the `TraceLevel` and `TraceLog` directives. These directives can be part of a connection string (in the case where `SQLDriverConnect` is being used) or within a DSN entry. The `TraceLog` is a log file where the traced output for the connection goes while the `TraceLevel` is the amount of tracing information wanted. The trace levels are:

- **NONE** No tracing information is printed.
- **MINIMAL** Routine name and parameters are included in the output.

- **LOW** In addition to the above, return values are included in the output.
- **MEDIUM** In addition to the above, the date and time of execution are included in the output.
- **HIGH** In addition to the above, parameter types are included in the output.

Also, third-party ODBC driver managers for Unix are available. Consult the documentation that accompanies these driver managers for information about their use.

#### See also

- [“The unixODBC driver manager” on page 353](#)
- [“UTF-32 ODBC driver managers for Unix” on page 354](#)

## The unixODBC driver manager

Versions of the unixODBC release before version 2.2.14 have incorrectly implemented some aspects of the 64-bit ODBC specification as defined by Microsoft. These differences will cause problems when using the unixODBC driver manager with the SQL Anywhere 64-bit ODBC driver.

To avoid these problems, you should be aware of the differences. One of them is the definition of `SQLLEN` and `SQLULEN`. These are 64-bit types in the Microsoft 64-bit ODBC specification, and are expected to be 64-bit quantities by the SQL Anywhere 64-bit ODBC driver. Some implementations of unixODBC define these two types as 32-bit quantities and this will result in problems when interfacing to the SQL Anywhere 64-bit ODBC driver.

There are three things that you must do to avoid problems on 64-bit platforms.

1. Instead of including the unixODBC headers like `sql.h` and `sqlext.h`, you should include the SQL Anywhere ODBC header file `unixodbc.h`. This will guarantee that you have the correct definitions for `SQLLEN` and `SQLULEN`. The header files in unixODBC 2.2.14 or later versions correct this problem.
2. You must ensure that you have used the correct types for all parameters. Use of the correct header file and the strong type checking of your C/C++ compiler should help in this area. You must also ensure that you have used the correct types for all variables that are set by the SQL Anywhere driver indirectly through pointers.
3. Do not use versions of the unixODBC driver manager before release 2.2.14. Link directly to the SQL Anywhere ODBC driver instead. For example, ensure that the `libodbc` shared object is linked to the SQL Anywhere driver.

```
libodbc.so.1 -> libdbodbc12_r.so.1
```

Alternatively, you can use the SQL Anywhere driver manager on platforms where it is available.

**See also**

- [“The SQL Anywhere ODBC driver manager for Unix” on page 352](#)
- [“Linking ODBC applications on Unix” on page 351](#)
- [“64-bit ODBC considerations” on page 368](#)

## UTF-32 ODBC driver managers for Unix

Versions of ODBC driver managers that define SQLWCHAR as 32-bit (UTF-32) quantities cannot be used with the SQL Anywhere ODBC driver that supports wide calls since this driver is built for 16-bit SQLWCHAR. For these cases, an ANSI-only version of the SQL Anywhere ODBC driver is provided. This version of the ODBC driver does not support the wide call interface (for example, SQLConnectW).

The shared object name of the driver is *libdbodbcansi12\_r*. Only a threaded variant of the driver is provided. On Mac OS X, in addition to the dylib, the driver is also available in bundle form (*dbodbcansi12\_r.bundle*). Certain frameworks, such as Real Basic, do not work with the dylib and require the bundle.

The regular ODBC driver treats SQLWCHAR strings as UTF-16 strings. This driver can not be used with some ODBC driver managers, such as iODBC, which treat SQLWCHAR strings as UTF-32 strings. When dealing with Unicode-enabled drivers, these driver managers translate narrow calls from the application to wide calls into the driver. An ANSI-only driver gets around this behavior, allowing the driver to be used with such driver managers, as long as the application does not make any wide calls. Wide calls through iODBC, or any other driver manager with similar semantics, remain unsupported.

## ODBC samples

Several ODBC samples are included with SQL Anywhere. You can find the samples in the `%SQLANYSAMP12%\SQLAnywhere` subdirectories.

The samples in directories starting with ODBC illustrate separate and simple ODBC tasks, such as connecting to a database and executing statements. A complete sample ODBC program is supplied in `%SQLANYSAMP12%\SQLAnywhere\C\odbc.c`. This program performs the same actions as the embedded SQL dynamic cursor example program that is in the same directory.

**See also**

- [“Sample embedded SQL programs” on page 445](#)

## Building the sample ODBC program

The ODBC sample program is located in `%SQLANYSAMP12%\SQLAnywhere\C`. A batch file (shell script for Unix) is included that can be used to compile and link all the sample applications located in this folder.

### Build the sample ODBC program

1. Open a command prompt and change the directory to the `%SQLANYSAMPI2%\SQLAnywhere\C` directory.
2. Run the `build.bat` or `build64.bat` batch file.

For x64 platform builds, you may need to set up the correct environment for compiling and linking. Here is an example that builds the sample programs for an x64 platform.

```
set mssdk=c:\mssdk\v7.0
build64
```

### Build the sample ODBC program for Unix

1. Open a command shell and change the directory to the `samples-dir/SQLAnywhere/C` directory.
2. Run the `build.sh` shell script.

## Running the sample ODBC program

### Run the ODBC sample

1. Start the program:
  - For 32-bit Windows, run the file `%SQLANYSAMPI2%\SQLAnywhere\C\odbcwin.exe`.
  - For 64-bit Windows, run the file `%SQLANYSAMPI2%\SQLAnywhere\C\odbcx64.exe`.
  - For Unix, run the file `samples-dir/SQLAnywhere/C/odbc`.
2. Choose a table:
  - Choose one of the tables in the sample database. For example, you can enter **Customers** or **Employees**.

## 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.

The following handles are used in essentially all ODBC applications:

- **Environment** The environment handle provides a global context in which to access data. Every ODBC application must allocate exactly one environment handle upon starting, and must free it at the end.

The following code illustrates how to allocate an environment handle:

```
SQLHENV env;  
SQLRETURN rc;
```

```
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL
    _NULL_HANDLE, &env );
```

- **Connection** A connection is specified by an ODBC driver and a data source. 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.

The following code illustrates how to allocate a connection handle:

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

- **Statement** A statement handle 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 both for cursor operations (fetching data) and for single statement execution (for example, INSERT, UPDATE, and DELETE).

The following code illustrates how to allocate a statement handle:

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

## ODBC handles

The handle types required for ODBC programs are as follows:

Item	Handle type
Environment	SQLHENV
Connection	SQLHDBC
Statement	SQLHSTMT
Descriptor	SQLHDESC

### Use an ODBC handle

1. Call the SQLAllocHandle function.

SQLAllocHandle takes the following 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 information, see `SQLAllocHandle` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms712455.aspx>.

2. Use the handle in subsequent function calls.
3. Free the object using `SQLFreeHandle`.

`SQLFreeHandle` takes the following parameters:

- an identifier for the type of item being freed
- the handle of the item being freed

For information, see `SQLFreeHandle` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms710123.aspx>.

### Example

The following 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
}
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

### See also

- “Error handling in ODBC” on page 382

## ODBC example

A simple ODBC program that connects to the SQL Anywhere sample database and immediately disconnects can be found in `%SQLANYSAMPI2%\SQLAnywhere\ODBCConnect\odbcconnect.cpp`. This example shows the steps required in setting up the environment to make a connection to a database server, as well as the steps required in disconnecting from the server and freeing up resources.

## ODBC connection functions

ODBC supplies a set of connection functions. Which one you use depends on how you expect your application to be deployed and used:

- **SQLConnect** The simplest connection function.

`SQLConnect` takes a data source name and optional user ID and password. You may want to use `SQLConnect` if you hard-code a data source name into your application.

For more information, see `SQLConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms711810.aspx>.

- **SQLDriverConnect** Connects to a data source using a connection string.

`SQLDriverConnect` allows the application to use SQL Anywhere-specific connection information that is external to the data source. Also, you can use `SQLDriverConnect` to request that the SQL Anywhere driver prompt for connection information.

`SQLDriverConnect` can also be used to connect without specifying a data source. The SQL Anywhere ODBC driver name is specified instead. The following example connects to a server and database that is already running.

```
SQLSMALLINT cso;
SQLCHAR     scso[2048];

SQLDriverConnect( hdbc, NULL,
                 "Driver=SQL Anywhere 12;UID=DBA;PWD=sql", SQL_NTS,
                 scso, sizeof(scso)-1,
                 &cso, SQL_DRIVER_NOPROMPT );
```

For more information, see `SQLDriverConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms715433.aspx>.

- **SQLBrowseConnect** Connects to a data source using a connection string, like `SQLDriverConnect`.

`SQLBrowseConnect` allows your application to build its own windows to prompt for connection information and to browse for data sources used by a particular driver (in this case the SQL Anywhere driver).

For more information, see `SQLBrowseConnect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms714565.aspx>.

### See also

- “Connection parameters” [*SQL Anywhere Server - Database Administration*]

## Establishing a connection

Your application must establish a connection before it can perform any database operations.

### Establish an ODBC connection

1. Allocate an ODBC environment.

For example:

```
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. Allocate an ODBC connection item.

For example:

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

4. Set any connection attributes that must be set before connecting.

Some connection attributes must be set before establishing a connection or after establishing a connection, while others can be set either before or after. The `SQL_AUTOCOMMIT` attribute is one that can be set before or after:

```
retcode = SQLSetConnectAttr( dbc,
    SQL_AUTOCOMMIT,
    (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
```

5. If necessary, assemble the data source or connection string.

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

6. Call the ODBC connection function.

For example:

```
if (retcode == SQL_SUCCESS
    || retcode == SQL_SUCCESS_WITH_INFO) {
    printf( "dbc allocated\n" );
    retcode = SQLConnect( dbc,
        (SQLCHAR*) "SQL Anywhere 12 Demo", SQL_NTS,
        (SQLCHAR* ) "DBA", SQL_NTS,
        (SQLCHAR*) "sql", SQL_NTS );
    if (retcode == SQL_SUCCESS
        || retcode == SQL_SUCCESS_WITH_INFO){
        // successfully connected.
    }
}
```

You can find a complete sample in `%SQLANYSAMPI2%\SQLAnywhere\ODBCConnect\odbcconnect.cpp`.

## Notes

- **SQL\_NTS** 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`).
- **SQLSetConnectAttr** By default, ODBC operates in autocommit mode. This mode is turned off by setting `SQL_AUTOCOMMIT` to false.

**See also**

- “Setting connection attributes” on page 360

## Setting connection attributes

You use the `SQLSetConnectAttr` function to control details of the connection. For example, the following statement turns off ODBC autocommit behavior.

```
retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,  
                             (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
```

For more information, see `SQLSetConnectAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms713605.aspx>.

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

**See also**

- “Connection parameters” [*SQL Anywhere Server - Database Administration*]

## Getting connection attributes

You use the `SQLGetConnectAttr` function to get details of the connection. For example, the following statement returns the connection state.

```
retcode = SQLGetConnectAttr( dbc, SQL_ATTR_CONNECTION_DEAD,  
                             (SQLPOINTER)&closed, SQL_IS_INTEGER, 0 );
```

When using the `SQLGetConnectAttr` function to get the `SQL_ATTR_CONNECTION_DEAD` attribute, the value `SQL_CD_TRUE` is returned if the connection has been dropped even if no request has been sent to the server since the connection was dropped. Determining if the connection has been dropped is done without making a request to the server, and the dropped connection is detected within a few seconds. The connection can be dropped for several reasons, such as an idle timeout.

For more information, including a list of connection attributes, see `SQLGetConnectAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms710297.aspx>.

## Threads and connections in ODBC applications

You can develop multithreaded ODBC applications for SQL Anywhere. It is recommended that you use a separate connection for each thread.

You can use a single connection for multiple threads. However, the database server does not allow more than one active request for any one connection at a time. If one thread executes a statement that takes a long time, all other threads must wait until the request is complete.

## Server options changed by ODBC

The SQL Anywhere ODBC driver sets some temporary server options when connecting to a SQL Anywhere database. The following options are set as indicated.

- **date\_format** yyyy-mm-dd
- **date\_order** ymd
- **isolation\_level** based on the SQL\_ATTR\_TXN\_ISOLATION/  
SA\_SQL\_ATTR\_TXN\_ISOLATION attribute setting of SQLSetConnectAttr. The following options are available.

```
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SA_SQL_TXN_SNAPSHOT
SA_SQL_TXN_STATEMENT_SNAPSHOT
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
```

- **time\_format** hh:nn:ss
- **timestamp\_format** yyyy-mm-dd hh:nn:ss.ssssss
- **timestamp\_with\_time\_zone\_format** yyyy-mm-dd hh:nn:ss.ssssss +hh:nn

To restore the default option setting, execute a SET statement. Here is an example of a statement that will reset the timestamp\_format option.

```
set temporary option timestamp_format =
```

### See also

- [“ODBC transaction isolation levels” on page 374](#)

## SQLSetConnectAttr extended connection attributes

The SQL Anywhere ODBC driver supports some extended connection attributes.

- **SA\_REGISTER\_MESSAGE\_CALLBACK** Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The message handler callback prototype is as follows:

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char msg_type,
long code,
```

```
unsigned short length,  
char * message  
);
```

The following possible values for *msg\_type* are defined in *sqldef.h*.

- **MESSAGE\_TYPE\_INFO** The message type was INFO.
- **MESSAGE\_TYPE\_WARNING** The message type was WARNING.
- **MESSAGE\_TYPE\_ACTION** The message type was ACTION.
- **MESSAGE\_TYPE\_STATUS** The message type was STATUS.
- **MESSAGE\_TYPE\_PROGRESS** The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

A SQLCODE associated with the message may be provided in *icode*. When not available, the *code* parameter value is 0.

The length of the message is contained in *length*.

A pointer to the message is contained in *message*. Note that *message* is not null-terminated. Your application must be designed to handle this. The following is an example.

```
memcpy( mybuff, msg, len );  
mybuff[ len ] = '\\0';
```

To register the message handler in ODBC, call the `SQLSetConnectAttr` function as follows:

```
rc = SQLSetConnectAttr(  
    hdbc,  
    SA_REGISTER_MESSAGE_CALLBACK,  
    (SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

To unregister the message handler in ODBC, call the `SQLSetConnectAttr` function as follows:

```
rc = SQLSetConnectAttr(  
    hdbc,  
    SA_REGISTER_MESSAGE_CALLBACK,  
    NULL, SQL_IS_POINTER );
```

- **SA\_GET\_MESSAGE\_CALLBACK\_PARM** To retrieve the value of the SQLHDBC connection handle that will be passed to message handler callback routine, use `SQLGetConnectAttr` with the `SA_GET_MESSAGE_CALLBACK_PARM` parameter.

```
SQLHDBC callback_hdbc = NULL;  
rc = SQLGetConnectAttr(  
    hdbc,  
    SA_GET_MESSAGE_CALLBACK_PARM,  
    (SQLPOINTER) &callback_hdbc, 0, 0 );
```

The returned value will be the same as the parameter value that is passed to the message handler callback routine.

- **SA\_REGISTER\_VALIDATE\_FILE\_TRANSFER\_CALLBACK** This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the ODBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the ODBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(
void * sqlca,
char * file_name,
int is_write
);
```

The *file\_name* parameter is the name of the file to be read or written. The *is\_write* parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the ODBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. Note that if the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

- **SA\_SQL\_ATTR\_TXN\_ISOLATION** This is used to set an extended transaction isolation level. The following example sets a Snapshot isolation level:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

#### See also

- “MESSAGE statement” [[SQL Anywhere Server - SQL Reference](#)]
- “progress\_messages option” [[SQL Anywhere Server - Database Administration](#)]
- “ODBC transaction isolation levels” on page 374

## Calling ODBC functions from DllMain

Note that ODBC functions should not be called directly or indirectly from the DllMain function in a Windows Dynamic Link Library. The DllMain entry point function is intended to perform only simple

initialization and termination tasks. Calling ODBC functions like `SQLFreeHandle`, `SQLFreeConnect`, and `SQLFreeEnv` can create deadlocks and circular dependencies.

The following code example illustrates a bad programming practice. When the Microsoft ODBC Driver Manager detects that the last access to the SQL Anywhere ODBC driver has completed, it will do a driver unload. When the SQL Anywhere ODBC driver shuts down, it stops any active threads. Thread termination results in a recursive thread detach call into `DllMain`. Since the call into `DllMain` is serialized, and a call is underway, the new thread detach call will never get started. The SQL Anywhere ODBC driver will wait forever for its threads to terminate and your application will hang.

```
BOOL APIENTRY DllMain( HMODULE hinstDLL,
    DWORD fdwReason,
    LPVOID lpvReserved
)
{
    HANDLE *handles;
    switch( fdwReason ) {
        .
        .
        .
        case DLL_THREAD_DETACH:
            /* do thread cleanup */
            handles = (HANDLE *) TlsGetValue( TlsIndex );
            if( handles != NULL )
            {
                HENV     tls_henv;
                HDBC     tls_hdbc;

                tls_henv = (HENV) handles[0];
                tls_hdbc = (HDBC) handles[1];
                if( tls_hdbc != NULL )
                    SQLFreeHandle( SQL_HANDLE_DBC, tls_hdbc );
                if( tls_henv != NULL )
                    SQLFreeHandle( SQL_HANDLE_ENV, tls_henv );
                handles[0] = NULL;
                handles[1] = NULL;
            }
            break;
        .
        .
        .
    }
    return TRUE;          /* indicate success */
}
```

For more information, read the Microsoft white paper on Best Practices for Creating DLLs at <http://msdn.microsoft.com/en-us/windows/hardware/gg487379.aspx>.

## Ways to execute SQL statements

ODBC includes several functions for executing SQL statements:

- **Direct execution** SQL Anywhere 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 so improves performance.

### See also

- [“Executing prepared statements” on page 367](#)

## Executing statements directly

The `SQLExecDirect` function prepares and executes a SQL statement. The statement may include parameters.

The following code fragment illustrates how to execute a statement without parameters. The `SQLExecDirect` function takes a statement handle, a SQL string, and a length or termination indicator, which in this case is a null-terminated string indicator.

The procedure described in this section is straightforward but inflexible. The application cannot take any input from the user to modify the statement. For a more flexible method of constructing statements, see [“Executing statements with bound parameters” on page 365](#).

### Execute a SQL statement in an ODBC application

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

For example, the following statement allocates a handle of type `SQL_HANDLE_STMT` with name `stmt`, on a connection with handle `dbc`:

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

2. Call the `SQLExecDirect` function to execute the statement:

For example, the following lines declare a statement and execute it. The declaration of `deletestmt` would usually occur at the beginning of the function:

```
SQLCHAR deletestmt[ STMT_LEN ] =  
    "DELETE FROM Departments WHERE DepartmentID = 201";  
SQLExecDirect( stmt, deletestmt, SQL_NTS );
```

For a complete sample with error checking, see [%SQLANYSAMP12%\SQLAnywhere\ODBCExecute\odbcexecute.cpp](#).

For more information about `SQLExecDirect`, see `SQLExecDirect` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms713611.aspx>.

## Executing statements with bound parameters

This section describes how to construct and execute a SQL statement, using bound parameters to set values for statement parameters at runtime.

## Execute a SQL statement with bound parameters in an ODBC application

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

For example, the following statement allocates a handle of type `SQL_HANDLE_STMT` with name `stmt`, on a connection with handle `dbc`:

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

2. Bind parameters for the statement using `SQLBindParameter`.

For example, the following lines declare variables to hold the values for the department ID, department name, and manager ID, and for the statement string. They then bind parameters to the first, second, and third parameters of a statement executed using the `stmt` statement handle.

```
#defined DEPT_NAME_LEN 40
SQLLEN cbDeptID = 0,
      cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLCHAR insertstmt[ STMT_LEN ] =
  "INSERT INTO Departments "
  "( DepartmentID, DepartmentName, DepartmentHeadID )"
  "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, the following lines assign values to the parameters for the fragment of step 2.

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

Commonly, these variables would be set in response to user action.

4. Execute the statement using `SQLExecDirect`.

For example, the following line executes the statement string held in `insertstmt` on the statement handle `stmt`.

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

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

The above code fragments do not include error checking. For a complete sample, including error checking, see `%SQLANYSAMPI2%\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.



For more information about SQLExecDirect, see SQLExecDirect in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms713611.aspx>.

### See also

- “Executing prepared statements” on page 367

## Executing prepared statements

Prepared statements provide performance advantages for statements that are used repeatedly. ODBC provides a full set of functions for using prepared statements.

### Execute a prepared SQL statement

1. Prepare the statement using SQLPrepare.

For example, the following code fragment illustrates how to prepare an INSERT statement:

```
SQLRETURN  retcode;
SQLHSTMT   stmt;
retcode = SQLPrepare( stmt,
    "INSERT INTO Departments
      ( DepartmentID, DepartmentName, DepartmentHeadID )
      VALUES (?, ?, ?,)",
    SQL_NTS);
```

In this example:

- **retcode** Holds a return code that should be tested for success or failure of the operation.
  - **stmt** Provides a handle to the statement so that it can be referenced later.
  - **?** The question marks are placeholders for statement parameters. A placeholder is put in the statement to indicate where host variables are to be accessed. A placeholder is either a question mark (?) or a host variable reference (a host variable name preceded by a colon). In the latter case, the host variable name used in the actual text of the statement serves only as a placeholder indicating that a corresponding parameter will be bound to it. It need not match the actual parameter name.
2. Set statement parameter values using SQLBindParameter.

For example, the following function call sets the value of the DepartmentID variable:

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

In this example:

- **stmt** is the statement handle.
- **1** indicates that this call sets the value of the first placeholder.
- **SQL\_PARAM\_INPUT** indicates that the parameter is an input statement.
- **SQL\_C\_SHORT** indicates the C data type being used in the application.
- **SQL\_INTEGER** indicates SQL data type being used in the database.

The next two parameters indicate the column precision and the number of decimal digits: both zero for integers.

- **&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.

4. Execute the statement:

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

Steps 2 to 4 can be carried out multiple times.

5. Drop the statement.

Dropping the statement frees resources associated with the statement itself. You drop statements using `SQLFreeHandle`.

For a complete sample, including error checking, see `%SQLANYSAMP12%\SQLAnywhere\ODBCPrepare\odbcprepare.cpp`.

For more information about SQLPrepare, see SQLPrepare in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms710926.aspx>.

### See also

- “Prepared statements” on page 2

## 64-bit ODBC considerations

When you use an ODBC function like `SQLBindCol`, `SQLBindParameter`, or `SQLGetData`, some of the parameters are typed as `SQLLEN` or `SQLULEN` in the function prototype. Depending on the date of the Microsoft ODBC API Reference documentation that you are looking at, you might see the same parameters described as `SQLINTEGER` or `SQLUINTEGER`.

`SQLLEN` and `SQLULEN` data items are 64 bits in a 64-bit ODBC application and 32 bits in a 32-bit ODBC application. `SQLINTEGER` and `SQLUINTEGER` data items are 32 bits on all platforms.

To illustrate the problem, the following ODBC function prototype was excerpted from an older copy of the Microsoft ODBC API Reference.

```

SQLRETURN SQLGetData(
    SQLHSTMT StatementHandle,
    SQLUSMALLINT ColumnNumber,
    SQLSMALLINT TargetType,
    SQLPOINTER TargetValuePtr,
    SQLINTEGER BufferLength,
    SQLINTEGER *StrLen_or_IndPtr);

```

Compare this with the actual function prototype found in *sql.h* in Microsoft Visual Studio version 8.

```

SQLRETURN SQL_API SQLGetData(
    SQLHSTMT StatementHandle,
    SQLUSMALLINT ColumnNumber,
    SQLSMALLINT TargetType,
    SQLPOINTER TargetValue,
    SQLLEN BufferLength,
    SQLLEN *StrLen_or_Ind);

```

As you can see, the `BufferLength` and `StrLen_or_Ind` parameters are now typed as `SQLLEN`, not `SQLINTEGER`. For the 64-bit platform, these are 64-bit quantities, not 32-bit quantities as indicated in the Microsoft documentation.

To avoid issues with cross-platform compilation, SQL Anywhere provides its own ODBC header files. For Windows platforms, you should include the *ntodbc.h* header file. For Unix platforms such as Linux, you should include the *unixodbc.h* header file. Use of these header files ensures compatibility with the corresponding SQL Anywhere ODBC driver for the target platform.

The following table lists some common ODBC types that have the same or different storage sizes on 64-bit and 32-bit platforms.

ODBC API	64-bit platform	32-bit platform
SQLINTEGER	32 bits	32 bits
SQLUINTEGER	32 bits	32 bits
SQLLEN	64 bits	32 bits
SQLULEN	64 bits	32 bits
SQLSETPOSIROW	64 bits	16 bits
SQL_C_BOOKMARK	64 bits	32 bits
BOOKMARK	64 bits	32 bits

If you declare data variables and parameters incorrectly, then you may encounter incorrect software behavior.

The following table summarizes the ODBC API function prototypes that have changed with the introduction of 64-bit support. The parameters that are affected are noted. The parameter name as documented by Microsoft is shown in parentheses when it differs from the actual parameter name used in the function prototype. The parameter names are those used in the Microsoft Visual Studio version 8 header files.

<b>ODBC API</b>	<b>Parameter (documented parameter name)</b>
SQLBindCol	SQLLEN BufferLength SQLLEN *Strlen_or_Ind
SQLBindParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind
SQLBindParameter	SQLULEN cbColDef (ColumnSize) SQLLEN cbValueMax (BufferLength) SQLLEN *pcbValue (Strlen_or_IndPtr)
SQLColAttribute	SQLLEN *NumericAttribute
SQLColAttributes	SQLLEN *pfDesc
SQLDescribeCol	SQLULEN *ColumnSize (ColumnSizePtr)
SQLDescribeParam	SQLULEN *pcbParamDef (ParameterSizePtr)
SQLExtendedFetch	SQLLEN irow (FetchOffset) SQLULEN *pcrow (RowCountPtr)
SQLFetchScroll	SQLLEN FetchOffset
SQLGetData	SQLLEN BufferLength SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLGetDescRec	SQLLEN *Length (LengthPtr)
SQLParamOptions	SQLULEN crow, SQLULEN *pirow
SQLPutData	SQLLEN Strlen_or_Ind
SQLRowCount	SQLLEN *RowCount (RowCountPtr)
SQLSetConnectOption	SQLULEN Value

ODBC API	Parameter (documented parameter name)
SQLSetDescRec	SQLLEN Length SQLLEN *StringLength (StringLengthPtr) SQLLEN *Indicator (IndicatorPtr)
SQLSetParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLSetPos	SQLSETPOSIROW irow (RowNumber)
SQLSetScrollOptions	SQLLEN crowKeyset
SQLSetStmtOption	SQLULEN Value

Some values passed into and returned from ODBC API calls through pointers have changed to accommodate 64-bit applications. For example, the following values for the SQLSetStmtAttr and SQLSetDescField functions are no longer SQLINTEGER/SQLINTEGER. The same rule applies to the corresponding parameters for the SQLGetStmtAttr and SQLGetDescField functions.

ODBC API	Type for Value/ValuePtr variable
SQLSetStmtAttr(SQL_ATTR_FETCH_BOOKMARK_PTR)	SQLLEN * value
SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_LENGTH)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_PARAM_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMS_PROCESSED_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_ARRAY_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_ARRAY_SIZE)	SQLULEN value

ODBC API	Type for Value/ValuePtr variable
SQLSetDescField(SQL_DESC_BIND_OFFSET_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_ROWS_PROCESSED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_DISPLAY_SIZE)	SQLLEN value
SQLSetDescField(SQL_DESC_INDICATOR_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)	SQLLEN * value

Note that the current Microsoft ODBC API Reference for SQLSetConnectAttr/SQLGetConnectAttr describes the numeric attribute values as SQLUINTEGER. Note that the type of these attribute values differs from the SQLSetStmtAttr/SQLGetStmtAttr attribute values which are described as SQLULEN.

#### Caution

There is one connection attribute, `SQL_ATTR_ODBC_CURSORS`, that is handled by the Microsoft ODBC Driver Manager. Although the Microsoft ODBC API Reference says the attribute is a SQLUINTEGER value specifying how the Driver Manager uses the ODBC cursor library, the 64-bit version of the Driver Manager returns a 64-bit SQLULEN value for SQLGetConnectAttr.

```
__int64 datavalue = 0x1234567812345678;
rc = SQLGetConnectAttr( hdbc, attr, &datavalue, 0, 0 );
```

After the call to SQLGetConnectAttr, the value of `datavalue` is `0x0000000000000002` which demonstrates that a 64-bit value is stored. Care must be exercised to avoid this Microsoft Driver Manager bug. Other attribute values are handled by the SQL Anywhere ODBC driver which returns SQLUINTEGER values as described by the ODBC API Reference.

For more information, see the Microsoft article "ODBC 64-Bit API Changes in MDAC 2.7" at <http://support.microsoft.com/kb/298678>.

## Data alignment requirements

When you use SQLBindCol, SQLBindParameter, or SQLGetData, a C data type is specified for the column or parameter. On certain platforms, the storage (memory) provided for each column must be properly aligned to fetch or store a value of the specified type. The ODBC driver checks for proper data alignment. When an object is not properly aligned, the ODBC driver will issue an **"Invalid string or buffer length"** message (SQLSTATE HY090 or S1090).

The following table lists memory alignment requirements for processors such as Sun Sparc, Itanium-IA64, and ARM-based devices. The memory address of the data value must be a multiple of the indicated value.

<b>C data type</b>	<b>Alignment required</b>
SQL_C_CHAR	none
SQL_C_BINARY	none
SQL_C_GUID	none
SQL_C_BIT	none
SQL_C_STINYINT	none
SQL_C_UTINYINT	none
SQL_C_TINYINT	none
SQL_C_NUMERIC	none
SQL_C_DEFAULT	none
SQL_C_SSHORT	2
SQL_C_USHORT	2
SQL_C_SHORT	2
SQL_C_DATE	2
SQL_C_TIME	2
SQL_C_TIMESTAMP	2
SQL_C_TYPE_DATE	2
SQL_C_TYPE_TIME	2
SQL_C_TYPE_TIMESTAMP	2
SQL_C_WCHAR	2 (buffer size must be a multiple of 2 on all platforms)
SQL_C_SLONG	4
SQL_C_ULONG	4
SQL_C_LONG	4

C data type	Alignment required
SQL_C_FLOAT	4
SQL_C_DOUBLE	8 (4 for ARM)
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

The x86, x64, and PowerPC platforms do not require memory alignment. The x64 platform includes Advanced Micro Devices (AMD) AMD64 processors and Intel Extended Memory 64 Technology (EM64T) processors.

## Result sets in ODBC applications

ODBC applications use cursors to manipulate and update result sets. SQL Anywhere provides extensive support for different kinds of cursors and cursor operations.

### See also

- [“Working with cursors” on page 7](#)

## ODBC transaction isolation levels

You can use `SQLSetConnectAttr` to set the transaction isolation level for a connection. The characteristics that determine the transaction isolation level that SQL Anywhere provides include the following:

- **SQL\_TXN\_READ\_UNCOMMITTED** Set isolation level to 0. When this attribute value is set, it isolates any data read from changes by others and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read. This is the default value for isolation level.
- **SQL\_TXN\_READ\_COMMITTED** Set isolation level to 1. When this attribute value is set, it does not isolate data read from changes by others, and changes made by others can be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read.
- **SQL\_TXN\_REPEATABLE\_READ** Set isolation level to 2. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This supports a repeatable read.
- **SQL\_TXN\_SERIALIZABLE** Set isolation level to 3. When this attribute value is set, it isolates any data read from changes by others, and changes made by others cannot be seen. The re-execution of the read statement is not affected by others. This supports a repeatable read.



- **SA\_SQL\_TXN\_SNAPSHOT** Set isolation level to Snapshot. When this attribute value is set, it provides a single view of the database for the entire transaction.
- **SA\_SQL\_TXN\_STATEMENT\_SNAPSHOT** Set isolation level to Statement-snapshot. When this attribute value is set, it provides less consistency than Snapshot isolation, but may be useful when long running transactions result in too much space being used in the temporary file by the version store.
- **SA\_SQL\_TXN\_READONLY\_STATEMENT\_SNAPSHOT** Set isolation level to Readonly-statement-snapshot. When this attribute value is set, it provides less consistency than Statement-snapshot isolation, but avoids the possibility of update conflicts. Therefore, it is most appropriate for porting applications originally intended to run under different isolation levels.

Note that the `allow_snapshot_isolation` database option must be set to On to use the Snapshot, Statement-snapshot, or Readonly-statement-snapshot settings.

For more information, see `SQLSetConnectAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms713605.aspx>.

### Example

The following fragment sets the isolation level to Snapshot:

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
    SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

## ODBC cursor characteristics

ODBC functions that execute statements and manipulate result sets, use cursors to perform their tasks. Applications open a cursor implicitly whenever they execute a `SQLExecute` or `SQLExecDirect` function.

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 SQL Anywhere provides a cursor optimized for performance in this case.

For a simple example of a forward-only cursor, see “Data retrieval” on page 376.

For applications that need to scroll both forward and backward through a result set, such as many graphical user interface applications, cursor behavior is more complex. What does the application when it returns to a row that has been updated by some other application? ODBC defines a variety of **scrollable cursors** to allow you to build in the behavior that suits your application. SQL Anywhere provides a full set of cursors to match the ODBC scrollable cursor types.

You set the required ODBC cursor characteristics by calling the `SQLSetStmtAttr` function that defines statement attributes. You must call `SQLSetStmtAttr` before executing a statement that creates a result set.

You can use `SQLSetStmtAttr` to set many cursor characteristics. The characteristics that determine the cursor type that SQL Anywhere supplies include the following:

- **SQL\_ATTR\_CURSOR\_SCROLLABLE** Set to `SQL_SCROLLABLE` for a scrollable cursor and `SQL_NONSCROLLABLE` for a forward-only cursor. `SQL_NONSCROLLABLE` is the default.
- **SQL\_ATTR\_CONCURRENCY** Set to one of the following values:
  - **SQL\_CONCUR\_READ\_ONLY** Disallow updates. `SQL_CONCUR_READ_ONLY` is the default.
  - **SQL\_CONCUR\_LOCK** Use the lowest level of locking sufficient to ensure that the row can be updated.
  - **SQL\_CONCUR\_ROWVER** Use optimistic concurrency control, comparing row versions such as SQLBase ROWID or Sybase `TIMESTAMP`.
  - **SQL\_CONCUR\_VALUES** Use optimistic concurrency control, comparing values.

For more information, see `SQLSetStmtAttr` in the Microsoft *ODBC API Reference* at <http://msdn.microsoft.com/en-us/library/ms712631.aspx>.

### Example

The following fragment requests a read-only, scrollable cursor:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
                SQL_SCROLLABLE, SQL_IS_UINTEGER );
```

## Data retrieval

To retrieve rows from a database, you execute a `SELECT` statement using `SQLExecute` or `SQLExecDirect`. This opens a cursor on the statement.

You then use `SQLFetch` or `SQLFetchScroll` to fetch rows through the cursor. These functions fetch the next rowset of data from the result set and return data for all bound columns. Using `SQLFetchScroll`, rowsets can be specified at an absolute or relative position or by bookmark. `SQLFetchScroll` replaces the older `SQLExtendedFetch` from the ODBC 2.0 specification.

When an application frees the statement using `SQLFreeHandle`, it closes the cursor.

To fetch values from a cursor, your application can use 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`. As an alternative, you can set the `SQL_ATTR_MAX_LENGTH` statement attribute to a value large enough to hold the entire value for the column. The default value for `SQL_ATTR_MAX_LENGTH` is 256 KB.

The SQL Anywhere ODBC driver implements `SQL_ATTR_MAX_LENGTH` in a different way than intended by the ODBC specification. The intended meaning for `SQL_ATTR_MAX_LENGTH` is that it

be used as a mechanism to truncate large fetches. This might be done for a "preview" mode where only the first part of the data is displayed. For example, instead of transmitting a 4 MB blob from the server to the client application, only the first 500 bytes of it might be transmitted (by setting `SQL_ATTR_MAX_LENGTH` to 500). The SQL Anywhere ODBC driver does not support this implementation.

The following code fragment opens a cursor on a query and retrieves data through the cursor. Error checking has been omitted to make the example easier to read. The fragment is taken from a complete sample, which can be found in `%SQLANYSAMP12%\SQLAnywhere\ODBCSelect\odbcselect.cpp`.

```
SQLINTEGER cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN retcode;
SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
SQLSetEnvAttr( env,
               SQL_ATTR_ODBC_VERSION,
               (void*)SQL_OV_ODBC3, 0);
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLConnect( dbc,
            (SQLCHAR*) "SQL Anywhere 12 Demo", SQL_NTS,
            (SQLCHAR*) "DBA", SQL_NTS,
            (SQLCHAR*) "sql", SQL_NTS );
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLBindCol( stmt, 1,
            SQL_C_SSHORT, &deptID, 0, &cbDeptID);
SQLBindCol( stmt, 2,
            SQL_C_CHAR, deptName,
            sizeof(deptName), &cbDeptName);
SQLBindCol( stmt, 3,
            SQL_C_SSHORT, &managerID, 0, &cbManagerID);
SQLExecDirect( stmt, (SQLCHAR * )
"SELECT DepartmentID, DepartmentName, DepartmentHeadID FROM Departments "
"ORDER BY DepartmentID", SQL_NTS );
while( ( retcode = SQLFetch( stmt ) ) != SQL_NO_DATA ){
    printf( "%d %20s %d\n", deptID, deptName, managerID );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
SQLDisconnect( dbc );
SQLFreeHandle( SQL_HANDLE_DBC, dbc );
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

The number of row positions you can fetch in a cursor is governed by the size of an integer. You can fetch rows numbered up to number 2147483646, which is one less than the value that can be held in a 32-bit integer. When using negative numbers (rows from the end) you can fetch down to one more than the largest negative value that can be held in an integer.

## Updating and deleting rows through a cursor

The Microsoft *ODBC Programmer's Reference* suggests that you use `SELECT...FOR UPDATE` to indicate that a query is updatable using positioned operations. You do not need to use the `FOR UPDATE` clause in SQL Anywhere: `SELECT` statements are automatically updatable as long as the following conditions are met:

- The underlying query supports updates.

That is to say, as long as a data manipulation statement on the columns in the result is meaningful, then positioned data manipulation statements can be carried out on the cursor.

The `ansi_update_constraints` database option limits the type of queries that are updatable.

- The cursor type supports updates.

If you are using a read-only cursor, you cannot update the result set.

ODBC provides two alternatives for carrying out positioned updates and deletes:

- Use the `SQLSetPos` function.

Depending on the parameters supplied (`SQL_POSITION`, `SQL_REFRESH`, `SQL_UPDATE`, `SQL_DELETE`) `SQLSetPos` sets the cursor position and allows an application to refresh data, or update, or delete data in the result set.

This is the method to use with SQL Anywhere.

- Send positioned `UPDATE` and `DELETE` statements using `SQLExecute`. This method should not be used with SQL Anywhere.

#### See also

- “`ansi_update_constraints` option” [[SQL Anywhere Server - Database Administration](#)]

## Bookmarks

ODBC provides **bookmarks**, which are values used to identify rows in a cursor. SQL Anywhere supports bookmarks for value-sensitive and insensitive cursors. For example, this means that the ODBC cursor types `SQL_CURSOR_STATIC` and `SQL_CURSOR_KEYSET_DRIVEN` support bookmarks while cursor types `SQL_CURSOR_DYNAMIC` and `SQL_CURSOR_FORWARD_ONLY` do not.

Before ODBC 3.0, a database could specify only whether it supported bookmarks or not: there was no interface to provide this information for each cursor type. There was no way for a database server to indicate for what kind of cursor bookmarks were supported. For ODBC 2 applications, SQL Anywhere returns that it does support bookmarks. There is therefore nothing to prevent you from trying to use bookmarks with dynamic cursors; however, you should not use this combination.

## Calling stored procedures

This section describes how to create and call stored procedures and process the results from an ODBC application.

#### Procedures and result sets

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 `SQLExtendedFetch` just like any other cursor.

Parameters to procedures should be passed using parameter markers (question marks). Use `SQLBindParameter` to assign a storage area for each parameter marker, whether it is an INPUT, OUTPUT, or INOUT parameter.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure-defined result set. Therefore, ODBC does not always describe column names as defined in the `RESULT` clause of the stored procedure definition. To avoid this problem, you can use column aliases in your procedure result set cursor.

### Example 1

This example creates and calls a procedure that does not return a result set. The procedure takes one INOUT parameter, and increments its value. In the example, the variable `num_col` has the value zero, since the procedure does not return a result set. Error checking has been omitted to make the example easier to read.

```
HDBC dbc;
HSTMT stmt;
long I;
SWORD num_col;

/* Create a procedure */
SQLAllocStmt( dbc, &stmt );
SQLExecDirect( stmt,
    "CREATE PROCEDURE Increment( INOUT a INT )" \
    " BEGIN" \
    "   SET a = a + 1" \
    " END", SQL_NTS );

/* Call the procedure to increment 'I' */
I = 1;
SQLBindParameter( stmt, 1, SQL_C_LONG, SQL_INTEGER, 0,
    0, &I, NULL );
SQLExecDirect( stmt, "CALL Increment( ? )",
    SQL_NTS );
SQLNumResultCols( stmt, &num_col );
do_something( I );
```

### Example 2

This example calls a procedure that returns a result set. In the example, the variable `num_col` will have the value 2 since the procedure returns a result set with two columns. Again, error checking has been omitted to make the example easier to read.

```
HDBC dbc;
HSTMT stmt;
SWORD num_col;
RETCODE retcode;
char ID[ 10 ];
char Surname[ 20 ];

/* Create the procedure */
SQLExecDirect( stmt,
    "CREATE PROCEDURE employees()" \
    " RESULT( ID CHAR(10), Surname CHAR(20))" \
```

```
" BEGIN" \  
" SELECT EmployeeID, Surname FROM Employees" \  
" END", SQL_NTS );  
  
/* Call the procedure - print the results */  
SQLExecDirect( stmt, "CALL employees()", SQL_NTS );  
SQLNumResultCols( stmt, &num_col );  
SQLBindCol( stmt, 1, SQL_C_CHAR, &ID,  
            sizeof(ID), NULL );  
SQLBindCol( stmt, 2, SQL_C_CHAR, &Surname,  
            sizeof(Surname), NULL );  
  
for( ;; ) {  
    retcode = SQLFetch( stmt );  
    if( retcode == SQL_NO_DATA_FOUND ) {  
        retcode = SQLMoreResults( stmt );  
        if( retcode == SQL_NO_DATA_FOUND ) break;  
    } else {  
        do_something( ID, Surname );  
    }  
}
```

### See also

- “Stored procedures, triggers, batches, and user defined functions” [[SQL Anywhere Server - SQL Usage](#)]

## ODBC escape syntax

You can use ODBC escape syntax from any ODBC application. This escape syntax allows you to call a set of common functions regardless of the database management system you are using. The general form for the escape syntax is

{ **keyword** *parameters* }

The set of keywords includes the following:

- **{d date-string}** The date string is any date value accepted by SQL Anywhere.
- **{t time-string}** The time string is any time value accepted by SQL Anywhere.
- **{ts date-string time-string}** The date/time string is any timestamp value accepted by SQL Anywhere.
- **{guid uuid-string}** The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** The outer-join-expr is a valid OUTER JOIN expression accepted by SQL Anywhere.
- **{? = call func(p1,...)}** The function is any valid function call accepted by SQL Anywhere.
- **{call proc(p1,...)}** The procedure is any valid stored procedure call accepted by SQL Anywhere.

- **{fn func(p1,...)}** The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the ODBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE( ) }
```

The following tables list the functions that are supported by the SQL Anywhere ODBC driver.

### SQL Anywhere ODBC driver supported functions

Numeric functions	String functions	System functions	Time/date functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIME- STAMP
ATAN2	CHARAC- TER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND

Numeric functions	String functions	System functions	Time/date functions
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

The ODBC escape syntax is identical to the JDBC escape syntax. In Interactive SQL, which uses JDBC, the braces *must* be doubled. There must not be a space between successive braces: "{{" is acceptable, but "{ {" is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

For example, to obtain database properties with the sa\_db\_info procedure using SQL escape syntax, you would execute the following in Interactive SQL:

```
{{CALL sa_db_info( 0 ) }}
```

## Error handling in ODBC

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the SQLError 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 deprecated and replaced by the SQLGetDiagRec function.

Every ODBC function returns a SQLRETURN, which is one of the following status codes:

Status code	Description
SQL_SUCCESS	No error.
SQL_SUCCESS_WITH_INFO	The function completed, but a call to SQLError will indicate a warning.  The most common case for this status is that a value being returned is too long for the buffer provided by the application.



Status code	Description
SQL_ERROR	The function did not complete because of an error. Call <code>SQLError</code> to get more information about the problem.
SQL_INVALID_HANDLE	An invalid environment, connection, or statement handle was passed as a parameter.  This often happens if a handle is used after it has been freed, or if the handle is the null pointer.
SQL_NO_DATA_FOUND	There is no information available.  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.  This is an advanced feature described in the ODBC SDK documentation under <code>SQLParamData</code> and <code>SQLPutData</code> .

Every environment, connection, and statement handle can have one or more errors or warnings associated with it. Each call to `SQLError` or `SQLGetDiagRec` returns the information for one error and removes the information for that error. If you do not call `SQLError` or `SQLGetDiagRec` to remove all errors, the errors are removed on the next function call that passes the same handle as a parameter.

Each call to `SQLError` passes three handles for an environment, connection, and statement. The first call uses `SQL_NULL_HSTMT` to get the error associated with a connection. Similarly, a call with both `SQL_NULL_DBC` and `SQL_NULL_HSTMT` get any error associated with the environment handle.

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.

`SQLError` and `SQLGetDiagRec` return `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.

### Example 1

The following code fragment uses `SQLError` and return codes:

```

/* Declare required variables */
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN retcode;
UCHAR errmsg[100];
/* Code omitted here */
retcode = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt );
if( retcode == SQL_ERROR ){
    SQLError( env, dbc, SQL_NULL_HSTMT, NULL, NULL,
             errmsg, sizeof(errmsg), NULL );
}

```

```
    /* Assume that print_error is defined */
    print_error( "Allocation failed", errmsg );
    return;
}

/* Delete items for order 2015 */
retcode = SQLExecDirect( stmt,
    "DELETE FROM SalesOrderItems WHERE ID=2015",
    SQL_NTS );
if( retcode == SQL_ERROR ) {
    SQLError( env, dbc, stmt, NULL, NULL,
        errmsg, sizeof(errmsg), NULL );
    /* Assume that print_error is defined */
    print_error( "Failed to delete items", errmsg );
    return;
}
```

## Example 2

The following code fragment uses SQLGetDiagRec and return codes:

```
/* Declare required variables */
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN retcode;
SQLSMALLINT errmsglen;
SQLINTEGER errnative;
UCHAR errmsg[255];
UCHAR errstate[5];
/* Code omitted here */
retcode = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( retcode == SQL_ERROR ){
    SQLGetDiagRec( SQL_HANDLE_DBC, dbc, 1, errstate,
        &errnative, errmsg, sizeof(errmsg), &errmsglen);
    /* Assume that print_error is defined */
    print_error( "Allocation failed",
        errstate, errnative, errmsg );
    return;
}
/* Delete items for order 2015 */
retcode = SQLExecDirect( stmt,
    "DELETE FROM SalesOrderItems WHERE ID=2015",
    SQL_NTS );
if( retcode == SQL_ERROR ) {
    SQLGetDiagRec( SQL_HANDLE_STMT, stmt,
        recnum, errstate,
        &errnative, errmsg, sizeof(errmsg), &errmsglen);
    /* Assume that print_error is defined */
    print_error( "Failed to delete items",
        errstate, errnative, errmsg );
    return;
}
```

---

# Java in the database

SQL Anywhere provides a mechanism for executing Java classes from within the database server environment. Using Java methods in the database server provides powerful ways of adding programming logic to a database.

Java support in the database offers the following:

- Reuse Java components in the different layers of your application—client, middle-tier, or server—and use them wherever it makes the most sense to you. SQL Anywhere becomes a platform for distributed computing.
- Java provides a more powerful language than the SQL stored procedure language for building logic into the database.
- Java can be used in the database server without jeopardizing the integrity, security, or robustness of the database and the server.

## The SQLJ standard

Java in the database is based on the SQLJ Part 1 proposed standard (ANSI/INCITS 331.1-1999). SQLJ Part 1 provides specifications for calling Java static methods as SQL stored procedures and functions.

## Learning about Java in the database

The following table outlines the documentation regarding the use of Java in the database.

Title	Purpose
<a href="#">“Java in the database”</a> (this chapter)	Java concepts and how to apply them in SQL Anywhere.
<a href="#">“Creating a Java class for use with SQL Anywhere”</a>	Practical steps for using Java in the database.
<a href="#">“JDBC support”</a>	Accessing data from Java classes, including distributed computing.

The following table is a guide to which parts of the Java documentation apply to you, depending on your interests and background.

If you...	Consider reading...
are a Java developer who wants to just get started.	<a href="#">“Creating a Java class for use with SQL Anywhere”</a>
want to know the key features of Java in the database.	<a href="#">“Java in the database FAQ”</a>

<b>If you...</b>	<b>Consider reading...</b>
want to find out how to access data from Java.	<a href="#">“JDBC support”</a>

## Java in the database FAQ

This section describes the key features of Java in the database.

### What are the key features of Java in the database?

Detailed explanations of all the following points appear in later sections.

- **You can run Java in the database** An external Java VM runs Java code on behalf of the database server.
- **You can access data from Java** SQL Anywhere lets you access data from Java.
- **SQL is preserved** The use of Java does not alter the behavior of existing SQL statements or other aspects of non-Java relational database behavior.

### How do I store Java classes in the database?

Java is an object-oriented language, so its instructions (source code) come in the form of classes. To execute Java in a database, you write the Java instructions outside the database and compile them outside the database into compiled classes (**byte code**), which are binary files holding Java instructions.

You then install these compiled classes into a database. Once installed, you can execute these classes from the database server as a stored procedure. For example, the following statement creates the interface to a Java procedure:

```
CREATE PROCEDURE insertfix()  
EXTERNAL NAME 'JDBCExample.InsertFixed()'V  
LANGUAGE JAVA;
```

SQL Anywhere facilitates a runtime environment for Java classes, not a Java development environment. You need a Java development environment, such as the Sun Microsystems Java Development Kit, to write and compile Java. You also need a Java Runtime Environment to execute Java classes.

#### See also

- [“Installing Java classes into a database” on page 393](#)

## How does Java get executed in a database?

SQL Anywhere launches a Java VM. The Java VM interprets compiled Java instructions and runs them on behalf of the database server. The database server starts the Java VM automatically when needed: you do not have to take any explicit action to start or stop the Java VM.

The SQL request processor in the database server has been extended so it can call into the Java VM to execute Java instructions. It can also process requests from the Java VM to enable data access from Java.

## Why Java?

Java provides several features that make it ideal for use in the database:

- Thorough error checking at compile time.
- Built-in error handling with a well-defined error handling methodology.
- Built-in garbage collection (memory recovery).
- Elimination of many bug-prone programming techniques.
- Strong security features.
- Platform-independent execution of Java code.

## What platforms is Java in the database supported on?

Java in the database is supported on all Unix and Windows operating systems except Windows Mobile.

## How do I use Java and SQL together?

Java methods are declared as stored procedures, and can then be called just like SQL stored procedures.

You must create a stored procedure that runs your method. For example:

```
CREATE PROCEDURE javaproc()  
EXTERNAL NAME 'JDBCExample.MyMethod()'V  
LANGUAGE JAVA;
```

You can use many of the classes that are part of the Java API as included in the Sun Microsystems Java Development Kit. You can also use classes created and compiled by Java developers.

### See also

- [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#)

## How can I use my own Java classes in databases?

The Java language is more powerful than SQL and compiled classes can be called from client applications as easily and in the same way as stored procedures. Java classes can contain both information about the subject and some computational logic. For example, you could design, write, and compile Java code to create an Employees class complete with various methods that perform operations on an Employees table. You install your Java classes as objects into a database and write SQL cover functions or procedures to invoke the methods in the Java classes.

### See also

- [“Installing Java classes into a database” on page 393](#)

## Java error handling

Java error handling code is separate from the code for normal processing.

Errors generate an exception object representing the error. This is called **throwing an exception**. A thrown exception terminates a Java program unless it is caught and handled properly at some level of the application.

Both Java API classes and custom-created classes can throw exceptions. In fact, users can create their own exception classes that throw their own custom-created classes.

If there is no exception handler in the body of the method where the exception occurred, then the search for an exception handler continues up the call stack. If the top of the call stack is reached and no exception handler has been found, the default exception handler of the Java interpreter running the application is called and the program terminates.

In SQL Anywhere, if a SQL statement calls a Java method, and an unhandled exception is thrown, a SQL error is generated. The full text of the Java exception plus the Java stack trace is displayed in the server messages window.

## Creating a Java class for use with SQL Anywhere

The following sections describe the steps involved in creating Java methods and calling them from SQL. It shows you how to compile and install a Java class into the database to make it available for use in SQL Anywhere. It also shows you how to access the class and its members and methods from SQL statements.

The following sections assume that you have a Java Development Kit (JDK) installed, including the Java compiler (javac) and Java VM.

Source code and batch files for the sample are provided in *%SQLANYSAMP12%\SQLAnywhere\JavaInvoice*.

The first step to using Java in the database is to write the Java code and compile it.

### Create the class file

1. Go to the *JavaInvoice* folder containing the sample Java program.
2. Compile the file to create the file *Invoice.class*.

```
javac Invoice.java
```

The class is now compiled and ready to be installed into the database.

## Java VMs

The database server must be set up to locate a Java VM. Since you can specify different Java VMs for each database, the ALTER EXTERNAL ENVIRONMENT statement can be used to indicate the location (path) of the Java VM.

```
ALTER EXTERNAL ENVIRONMENT JAVA  
LOCATION 'c:\\jdk1.6.0_20\\jre\\bin\\java.exe';
```

If the location is not set, the database server searches for the location of the Java VM, as follows:

- Check the JAVA\_HOME environment variable.
- Check the JAVAHOME environment variable.
- Check the path.
- If the VM cannot be located, return an error.

#### Note

JAVA\_HOME and JAVAHOME are environment variables commonly created when installing a Java VM. If neither of these exist, you can create them manually, and point them to the root directory of your Java VM. However, this is not required if you use the ALTER EXTERNAL ENVIRONMENT statement.

### Specify the location of the Java VM (Interactive SQL)

1. Start Interactive SQL and connect to the database.
2. In the SQL Statements pane, type the following statement:

```
ALTER EXTERNAL ENVIRONMENT JAVA  
LOCATION 'path\\java.exe';
```

Here, *path* indicates the location of the Java VM (for example, *c:\\jdk1.6.0\_20\\jre\\bin*).

Use the *java\_vm\_options* option to specify any additional command line options that are required to start the Java VM.

```
SET OPTION PUBLIC.java_vm_options='java-options';
```

If you want to use JAVA in the database, but do not have a Java Runtime Environment (JRE) installed, you can install and use any Java JRE that you want to. Once installed, it is best to set the JAVA\_HOME or JAVAHOME environment variable to point to the root of the installed JRE. Note that most Java installers set one of these environment variables by default. Once a JRE is installed and JAVA\_HOME or JAVAHOME is set correctly, you should then be able to use Java in the database without performing any additional steps.

#### See also

- “ALTER EXTERNAL ENVIRONMENT statement” [[SQL Anywhere Server - SQL Reference](#)]
- “java\_vm\_options option” [[SQL Anywhere Server - Database Administration](#)]

## Installing the sample Java class

Java classes must be installed into a database before they can be used. You can install classes from Sybase Central or Interactive SQL.

#### Install the class to the SQL Anywhere sample database (Sybase Central)

1. Start Sybase Central and connect to the sample database.
2. In the left pane, expand the **External Environments** folder.
3. Click **Java**.
4. Click **File » New » Java Class**.
5. Click **Browse** and browse to the location of *Invoice.class*.
6. Click **Finish**.

#### Install the class to the SQL Anywhere sample database (Interactive SQL)

1. Start Interactive SQL and connect to the sample database.
2. In the **SQL Statements** pane of Interactive SQL, type the following statement:

```
INSTALL JAVA NEW  
FROM FILE 'path\\Invoice.class';
```

Here *path* is the location of your compiled class file.

3. Press F5 to execute the statement.

The class is now installed into the sample database.

#### Notes

- At this point, no Java in the database operations have taken place. The class has been installed into the database and is ready for use.



- Changes made to the class file from now on are *not* automatically reflected in the copy of the class in the database. You must update the classes in the database if you want the changes reflected.

### See also

- [“Installing Java classes into a database” on page 393](#)

## The CLASSPATH environment variable

The Java Runtime Environment and the JDK Java compiler use the CLASSPATH environment variable to locate classes referenced within Java code. A CLASSPATH variable provides the link between Java code and the actual file path or URL location of the classes being referenced. For example, `import java.io.*` allows all the classes in the `java.io` package to be referenced without a fully qualified name. Only the class name is required in the following Java code to use classes from the `java.io` package. The CLASSPATH environment variable on the system where the Java class declaration is to be compiled must include the location of the Java directory, the root of the `java.io` package.

### CLASSPATH used to install classes

The CLASSPATH variable can be used to locate a file during the installation of classes. For example, the following statement installs a user-created Java class to a database, but only specifies the name of the file, not its full path and name. (Note that this statement involves no Java operations.)

```
INSTALL JAVA NEW
FROM FILE 'Invoice.class';
```

If the file specified is in a directory or ZIP file specified by the CLASSPATH environmental variable, SQL Anywhere successfully locates the file and installs the class.

## Calling methods in the Java class

To access the Java methods in the class, you must create stored procedures or functions that act as wrappers for the methods in the class.

### Call a Java method using Interactive SQL

1. Create the following SQL stored procedure to call the `Invoice.main` method in the sample class:

```
CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )
EXTERNAL NAME 'Invoice.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

This stored procedure acts as a wrapper to the Java method.

2. Call the stored procedure to call the Java method:

```
CALL InvoiceMain('to you');
```

If you examine the database server message log, you see the message "Hello to you" written there. The database server has redirected the output there from System.out.

Here are more examples of how to call Java methods, pass arguments, and return values.

### Create stored procedures/functions for the methods in the Invoice class

1. Create the following SQL stored procedures to pass arguments to and retrieve return values from the Java methods in the Invoice class:

```
-- Invoice.init takes a string argument (Ljava/lang/String;)
-- a double (D), a string argument (Ljava/lang/String;), and
-- another double (D), and returns nothing (V)
CREATE PROCEDURE init( IN arg1 CHAR(50),
                      IN arg2 DOUBLE,
                      IN arg3 CHAR(50),
                      IN arg4 DOUBLE)
EXTERNAL NAME 'Invoice.init(Ljava/lang/String;DLjava/lang/String;D)V'
LANGUAGE JAVA;

-- The Java methods below take no arguments and return a double (D)
-- or a string (Ljava/lang/String;)

CREATE FUNCTION rateOfTaxation()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.rateOfTaxation()D'
LANGUAGE JAVA;

CREATE FUNCTION totalSum()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.totalSum()D'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem1Description()
RETURNS CHAR(50)
EXTERNAL NAME 'Invoice.getLineItem1Description()Ljava/lang/String;'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem1Cost()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.getLineItem1Cost()D'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem2Description()
RETURNS CHAR(50)
EXTERNAL NAME 'Invoice.getLineItem2Description()Ljava/lang/String;'
LANGUAGE JAVA;

CREATE FUNCTION getLineItem2Cost()
RETURNS DOUBLE
EXTERNAL NAME 'Invoice.getLineItem2Cost()D'
LANGUAGE JAVA;
```

The descriptors for arguments to and return values from Java methods have the following meanings:

Field type	Java data type
B	byte

Field type	Java data type
C	char
D	double
F	float
I	int
J	long
L <i>class-name</i> ;	An instance of the class <i>class-name</i> . The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, <b>java/lang/String</b> .
S	short
V	void
Z	Boolean
[	Use one for each dimension of an array.

2. Call the stored procedure that is acting as a wrapper to call the Java method:

```
CALL init('Shirt',10.00,'Jacket',25.00);
SELECT getLineItem1Description() as Item1,
       getLineItem1Cost() as Item1Cost,
       getLineItem2Description() as Item2,
       getLineItem2Cost() as Item2Cost,
       rateOfTaxation() as TaxRate,
       totalSum() as Cost;
```

The query returns six columns with values as follows:

Item1	Item1Cost	Item2	Item2Cost	TaxRate	Cost
Shirt	10	Jacket	25	0.15	40.25

#### See also

- [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE FUNCTION statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Installing Java classes into a database

You can install Java classes into a database as:

- **A single class** You can install a single class into a database from a compiled class file. Class files typically have extension *.class*.
- **A JAR** You can install a set of classes all at once if they are in either a compressed or uncompressed JAR file. JAR files typically have the extension *.jar* or *.zip*. SQL Anywhere supports all compressed JAR files created with the JAR utility, and some other JAR compression schemes.

## Creating a class

Although the details of each step may differ depending on whether you are using a Java development tool, the steps involved in creating your own class generally include the following:

### Create a class

1. Define your class.

Write the Java code that defines your class. If you are using the JDK then you can use a text editor. If you are using a development tool, the development tool provides instructions.

**Note**

User classes must be 100% Java. Native methods are not allowed.

2. Name and save your class.

Save your class declaration (Java code) in a file with the extension *.java*. Make certain the name of the file is the same as the name of the class and that the case of both names is identical.

For example, a class called Utility should be saved in a file called *Utility.java*.

3. Compile your class.

This step turns your class declaration containing Java code into a new, separate file containing byte code. The name of the new file is the same as the Java code file, but has an extension of *.class*. You can run a compiled Java class in a Java Runtime Environment, regardless of the platform you compiled it on or the operating system of the runtime environment.

The JDK contains a Java compiler, *javac*.

## Installing a class

To make your Java class available within the database, you install the class into the database either from Sybase Central, or using the `INSTALL JAVA` statement from Interactive SQL or another application. You must know the path and file name of the class you want to install.

### Install a class (Sybase Central)

1. Connect to a database as a user with DBA authority.
2. Open the **External Environments** folder.
3. Under this folder, open the **Java** folder.
4. Right-click the right pane and click **New » Java Class**.
5. Follow the instructions in the wizard.

### Install a class (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute the following statement:

```
INSTALL JAVA NEW  
FROM FILE 'path\\ClassName.class';
```

*path* is the directory where the class file is located, and *ClassName.class* is the name of the class file.

The double backslash ensures that the backslash is not treated as an escape character.

For example, to install a class in a file named *Utility.class*, held in the directory *c:\source*, you would execute the following statement:

```
INSTALL JAVA NEW  
FROM FILE 'c:\\source\\Utility.class';
```

If you use a relative path, it must be relative to the current working directory of the database server.

### See also

- “INSTALL JAVA statement” [[SQL Anywhere Server - SQL Reference](#)]

## Installing a JAR

It is useful and common practice to collect sets of related classes together in packages, and to store one or more packages in a **JAR file**.

You install a JAR file the same way as you install a class file. A JAR file can have the extension JAR or ZIP. Each JAR file must have a name in the database. Usually, you use the same name as the JAR file, without the extension. For example, if you install a JAR file named *myjar.zip*, you would generally give it a JAR name of *myjar*.

### Install a JAR (Sybase Central)

1. Connect to the database as a user with DBA authority.

2. Open the **External Environments** folder.
3. Under this folder, open the **Java** folder.
4. Right-click the right pane and click **New » JAR File**.
5. Follow the instructions in the wizard.

### Install a JAR (SQL)

1. Connect to a database as a user with DBA authority.
2. Execute the following statement:

```
INSTALL JAVA NEW  
JAR 'jarname'  
FROM FILE 'path\\JarName.jar';
```

### See also

- [“INSTALL JAVA statement” \[SQL Anywhere Server - SQL Reference\]](#)

## Updates to classes and JAR files

You can update classes and JAR files using Sybase Central or by executing an `INSTALL JAVA` statement from Interactive SQL or some other client application.

To update a class or JAR, you must have DBA authority and a newer version of the compiled class file or JAR file available in a file on disk.

### When updated classes take effect

Only new connections established after installing the class, or that use the class for the first time after installing the class, use the new definition. Once the Java VM loads a class definition, it stays in memory until the connection closes.

If you have been using a Java class or objects based on a class in the current connection, you need to disconnect and reconnect to use the new class definition.

## Updating classes and JAR files

### Update a class or JAR (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the **External Environments** folder.
3. Under this folder, open the **Java** folder.
4. Locate the subfolder containing the class or JAR file you want to update.

5. Click the class or JAR file and then click **File » Update**.
6. In the **Update** window, specify the location and name of the class or JAR file to be updated. You can click **Browse** to search for it.

**Note**

You can also update a Java class or JAR file by right-clicking the class or JAR file name and choosing **Update**.

As well, you can update a Java class or JAR file by clicking **Update Now** on the **General** tab of its **Properties** window.

**Update a class or JAR (SQL)**

1. Connect to a database as a user with DBA authority.
2. Execute the following statement:

```
INSTALL JAVA UPDATE
[ JAR 'jarname' ]
FROM FILE 'filename';
```

If you are updating a JAR, you must enter the name by which the JAR is known in the database.

**See also**

- “[INSTALL JAVA statement](#)” [[SQL Anywhere Server - SQL Reference](#)]

## Special features of Java classes in the database

This section describes features of Java classes when used in the database.

### Calling the main method

You typically start Java applications (outside the database) by running the Java VM on a class that has a main method.

For example, the Invoice class in the file `%SQLANYSAMPI2%\SQLAnywhere\JavaInvoice\Invoice.java` has a main method. When you execute the class from the command line using a command such as the following, it is the main method that executes.

```
java Invoice
```

**Call the main method of a class from SQL**

1. Declare the method with an array of strings as an argument:

```
public static void main( java.lang.String args[] )
{
```

```
} ...
```

2. Create a stored procedure that wraps this method.

```
CREATE PROCEDURE JavaMain( in arg char(50) )  
EXTERNAL NAME 'JavaClass.main([Ljava/lang/String;)V'  
LANGUAGE JAVA;
```

3. Invoke the main method using the CALL statement.

```
call JavaMain( 'Hello world' );
```

Due to the limitations of the SQL language, only a single string can be passed.

### See also

- [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Threads in Java applications

With features of the `java.lang.Thread` package, you can use multiple threads in a Java application.

You can synchronize, suspend, resume, interrupt, or stop threads in Java applications.

## No Such Method Exception

If you supply an incorrect number of arguments when calling a Java method, or if you use an incorrect data type, the Java VM responds with a `java.lang.NoSuchMethodException` error. You should check the number and type of arguments.

### See also

- [“Calling methods in the Java class” on page 391](#)

## Returning result sets from Java methods

This section describes how to make result sets available from Java methods. You must write a Java method that returns a result set to the calling environment, and wrap this method in a SQL stored procedure declared to be `EXTERNAL NAME` of `LANGUAGE JAVA`.

### Return result sets from a Java method

1. Ensure that the Java method is declared as `public` and `static` in a public class.
2. For each result set you expect the method to return, ensure that the method has a parameter of type `java.sql.ResultSet[]`. These result set parameters must all occur at the end of the parameter list.



3. In the method, first create an instance of `java.sql.ResultSet` and then assign it to one of the `ResultSet[]` parameters.
4. Create a SQL stored procedure of type `EXTERNAL NAME LANGUAGE JAVA`. This type of procedure is a wrapper around a Java method. You can use a cursor on the SQL procedure result set in the same way as any other procedure that returns result sets.

For more information about the syntax for stored procedures that are wrappers for Java methods, see [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#).

### Example

The following simple class has a single method that executes a query and passes the result set back to the calling environment.

```
import java.sql.*;

public class MyResultSet
{
    public static void return_rset( ResultSet[] rset1 )
        throws SQLException
    {
        Connection conn = DriverManager.getConnection(
            "jdbc:default:connection" );
        Statement stmt = conn.createStatement();
        ResultSet rset =
            stmt.executeQuery (
                "SELECT Surname " +
                "FROM Customers" );
        rset1[0] = rset;
    }
}
```

You can expose the result set using a `CREATE PROCEDURE` statement that indicates the number of result sets returned from the procedure and the signature of the Java method.

A `CREATE PROCEDURE` statement indicating a result set could be defined as follows:

```
CREATE PROCEDURE result_set()
    RESULT (SurName person_name_t)
    DYNAMIC RESULT SETS 1
    EXTERNAL NAME
        'MyResultSet.return_rset([Ljava/sql/ResultSet; )V'
    LANGUAGE JAVA
```

You can open a cursor on this procedure, just as you can with any SQL Anywhere procedure returning result sets.

The string `( [Ljava/sql/ResultSet; )V` is a Java method signature that is a compact character representation of the number and type of the parameters and return value.

### See also

- [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Returning result sets” on page 425](#)

## Values returned from Java via stored procedures

You can use stored procedures created using the EXTERNAL NAME LANGUAGE JAVA as wrappers around Java methods. This section describes how to write your Java method to exploit OUT or INOUT parameters in the stored procedure.

Java does not have explicit support for INOUT or OUT parameters. Instead, you can use an array of the parameter. For example, to use an integer OUT parameter, create an array of exactly one integer:

```
public class Invoice
{
    public static boolean testOut( int[] param )
    {
        param[0] = 123;
        return true;
    }
}
```

The following procedure uses the testOut method:

```
CREATE PROCEDURE testOut( OUT p INTEGER )
EXTERNAL NAME 'Invoice.testOut([I]Z'
LANGUAGE JAVA;
```

The string ( [ I ] Z is a Java method signature, indicating that the method has a single parameter, which is an array of integers, and returns a Boolean value. You must define the method so that the method parameter you want to use as an OUT or INOUT parameter is an array of a Java data type that corresponds to the SQL data type of the OUT or INOUT parameter.

To test this, call the stored procedure with an uninitialized variable.

```
CREATE VARIABLE zap INTEGER;
CALL testOut( zap );
SELECT zap;
```

The result set is 123.

For more information about the syntax, including the method signature, see [“CREATE PROCEDURE statement \[External call\]” \[SQL Anywhere Server - SQL Reference\]](#).

## Security management for Java

Java provides security managers that you can use to control user access to security-sensitive features of your applications, such as file access and network access. You should take advantage of the security management features supported by your Java VM.

## Starting and stopping the Java VM

The Java VM loads automatically whenever the first Java operation is carried out. If you want to load it explicitly in readiness for carrying out Java operations, you can do so by executing the following statement:

```
START JAVA;
```

You can unload the Java VM when Java is not in use using the STOP JAVA statement. Only a user with DBA authority can execute this statement. The syntax is:

```
STOP JAVA;
```

## Shutdown hooks in the Java VM

The SQL Anywhere Java VM ClassLoader which is used in providing JAVA in the database support allows applications to install shutdown hooks. These shutdown hooks are very similar to the shutdown hooks that applications install with the JVM Runtime. When a connection that is using JAVA in the database support executes a STOP JAVA statement or disconnects, the ClassLoader for that connection will now run all shutdown hooks that have been installed for that particular connection prior to unloading. It should be noted that for regular JAVA in the database applications that install all Java classes within the database, the installation of shutdown hooks should not be necessary. The ClassLoader shutdown hooks should be used with extreme caution and should only be used to clean up any system wide resources that were allocated for the particular connection that is stopping Java. Also note that jdbc:default JDBC requests are not allowed within shutdown hooks since the jdbc:default connection is already closed prior to the ClassLoader shutdown hook being called.

To install a shutdown hook with the SQL Anywhere Java VM ClassLoader, an application must include *sajvm.jar* in the Java compiler classpath and it needs to execute code similar to the following:

```
SDHookThread hook = new SDHookThread( ... );  
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();  
((iAnywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook );
```

Note that the SDHookThread class extends the standard Thread class and that the above code must be executed by a class that was loaded by the ClassLoader for the current connection. Any class that is installed within the database and that is subsequently called via an external environment call will automatically be executed by the correct SQL Anywhere Java VM ClassLoader.

To remove a shutdown hook from the SQL Anywhere Java VM ClassLoader list, an application will need to execute code similar to the following:

```
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();  
((iAnywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( hook );
```

The above code must be executed by a class that was loaded by the ClassLoader for the current connection.

---

---

## JDBC support

JDBC is a call-level interface for Java applications. Developed by Sun Microsystems, JDBC provides you with a uniform interface to a wide range of relational databases, and provides a common base on which higher level tools and interfaces can be built. JDBC is now a standard part of Java and is included in the JDK.

SQL Anywhere includes JDBC 3.0 and 4.0 drivers, which are both Type 2 drivers.

SQL Anywhere also supports a pure Java JDBC driver, named jConnect, which is available from Sybase.

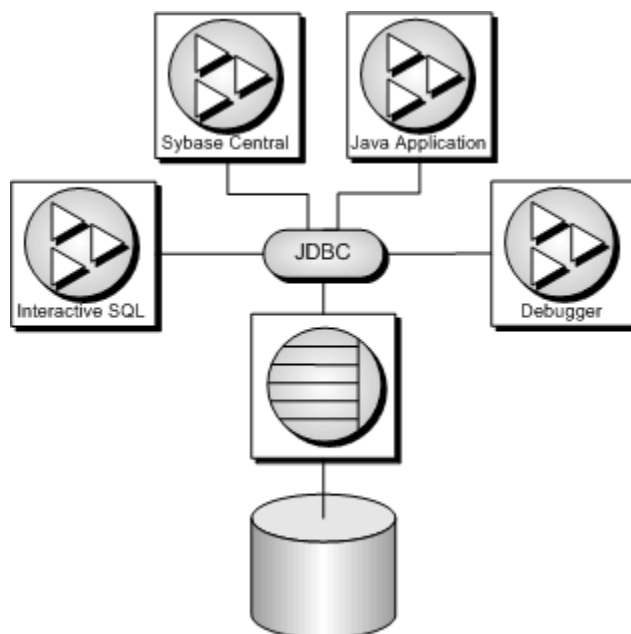
In addition to using JDBC as a client-side application programming interface, you can also use JDBC inside the database server to access data by using Java in the database.

### See also

- [“JDBC drivers” on page 404](#)

## JDBC applications

You can develop Java applications that use the JDBC API to connect to SQL Anywhere. Several of the applications supplied with SQL Anywhere use JDBC, such as the debugger, Sybase Central, and Interactive SQL.



Java and JDBC are also important programming languages for developing UltraLite applications.

JDBC can be used both from client applications and inside the database. Java classes using JDBC provide a more powerful alternative to SQL stored procedures for incorporating programming logic into the database.

JDBC provides a SQL interface for Java applications: if you want to access relational data from Java, you do so using JDBC calls.

The phrase **client application** applies both to applications running on a user's computer and to logic running on a middle-tier application server.

The examples illustrate the distinctive features of using JDBC in SQL Anywhere. For more information about JDBC programming, see any JDBC programming book.

You can use JDBC with SQL Anywhere in the following ways:

- **JDBC on the client** Java client applications can make JDBC calls to SQL Anywhere. The connection takes place through a JDBC driver.

SQL Anywhere includes JDBC 3.0 and 4.0 drivers, which are Type 2 JDBC drivers, and also supports the jConnect driver for pure Java applications, which is a Type 4 JDBC driver.

- **JDBC in the database** Java classes installed into a database can make JDBC calls to access and modify data in the database using an internal JDBC driver.

### JDBC resources

- **Example source code** You can find source code for the examples in this section in the directory *%SQLANYAMP12%\SQLAnywhere\JDBC*.
- **JDBC Specification** You can find more information about the JDBC Data Access API at <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html>.
- **Required software** You need TCP/IP to use the jConnect driver.

The jConnect driver is available at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.

### See also

- [“The jConnect JDBC driver” on page 409](#)

## JDBC drivers

SQL Anywhere supports the following JDBC drivers:

- **SQL Anywhere JDBC 3.0 driver** This driver communicates with SQL Anywhere using the Command Sequence client/server protocol. Its behavior is consistent with ODBC, embedded SQL, and OLE DB applications. The SQL Anywhere JDBC 3.0 driver can be used only with JRE 1.4 or

later. Note that the SQL Anywhere JDBC 4.0 driver is recommended for applications using JRE 1.6 or later.

- **SQL Anywhere JDBC 4.0 driver** This driver communicates with SQL Anywhere using the Command Sequence client/server protocol. Its behavior is consistent with ODBC, embedded SQL, and OLE DB applications. The SQL Anywhere JDBC 4.0 driver is the recommended JDBC driver for connecting to SQL Anywhere databases. The JDBC 4.0 driver can be used only with JRE 1.6 or later.

It is strongly recommended that applications using JRE 1.6 or later switch to the JDBC 4.0 driver instead of continuing to use the JDBC 3.0 driver. The JDBC 4.0 driver takes advantage of the new automatic JDBC driver registration. Hence, if an application wants to make use of the JDBC 4.0 driver, it no longer needs to perform a `Class.forName` call to get the JDBC driver loaded. It is instead sufficient to have the *sajdbc4.jar* file in the class file path and simply call `DriverManager.getConnection()` with a URL that begins with `jdbc:sqlanywhere`.

The JDBC 4.0 driver contains manifest information to allow it to be loaded as an OSGi (Open Services Gateway initiative) bundle.

With the JDBC 4.0 driver, metadata for NCHAR data now returns the column type as `java.sql.Types.NCHAR`, `NVARCHAR`, or `LONGNVARCHAR`. In addition, applications can now fetch NCHAR data using the `Get/SetNString` or `Get/SetNClob` methods instead of the `Get/SetString` and `Get/SetClob` methods.

- **jConnect** This driver is a 100% pure Java driver. It communicates with SQL Anywhere using the TDS client/server protocol.

jConnect and jConnect documentation are available at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.

When choosing which driver to use, you should consider the following factors:

- **Features** The SQL Anywhere JDBC 3.0 driver is JDBC 3.0 compliant. The SQL Anywhere JDBC 4.0 driver and jConnect are JDBC 4.0 compliant. Both versions of the SQL Anywhere JDBC driver provide fully-scrollable cursors when connected to a SQL Anywhere database. The jConnect JDBC driver provides scrollable cursors when connected to a SQL Anywhere database server, but the result set is cached on the client side. The jConnect JDBC driver provides fully-scrollable cursors when connected to an Adaptive Server Enterprise database.

The JDBC 3.0 and 4.0 API documentation is available at <http://www.oracle.com/technetwork/java/download-141179.html>. For a summary of the JDBC API methods supported by the SQL Anywhere JDBC driver, see “JDBC 3.0/4.0 API support” on page 434.

- **Pure Java** The jConnect driver is a pure Java solution. The SQL Anywhere JDBC drivers are based on the SQL Anywhere ODBC driver and are not pure Java solutions.
- **Performance** The SQL Anywhere JDBC drivers provide better performance for most purposes than the jConnect driver.
- **Compatibility** The TDS protocol used by the jConnect driver is shared with Adaptive Server Enterprise. Some aspects of the driver's behavior are governed by this protocol, and are configured to be compatible with Adaptive Server Enterprise.

For information about platform availability for the SQL Anywhere JDBC drivers and jConnect, see <http://www.sybase.com/detail?id=1061806>.

**See also**

- “jConnect on Windows Mobile” [*SQL Anywhere Server - Database Administration*]

## JDBC program structure

The following sequence of events typically occurs in JDBC applications:

1. **Create a Connection object** Calling a `getConnection` class method of the `DriverManager` class creates a `Connection` object, and establishes a connection with a database.
2. **Generate a Statement object** The `Connection` object generates a `Statement` object.
3. **Pass a SQL statement** A SQL statement that executes within the database environment is passed to the `Statement` object. If the statement is a query, this action returns a `ResultSet` object.

The `ResultSet` object contains the data returned from the SQL statement, but exposes it one row at a time (similar to the way a cursor works).

4. **Loop over the rows of the result set** The next method of the `ResultSet` object performs two actions:
  - The current row (the row in the result set exposed through the `ResultSet` object) advances one row.
  - A boolean value returns to indicate whether there is a row to advance to.
5. **For each row, retrieve the values** Values are retrieved for each column in the `ResultSet` object by identifying either the name or position of the column. You can use the `getData` method to get the value from a column on the current row.

Java objects can use JDBC objects to interact with a database and get data for their own use.

## Differences between client- and server-side JDBC connections

A difference between JDBC on the client and in the database server lies in establishing a connection with the database environment.

- **Client side** In client-side JDBC, establishing a connection requires a SQL Anywhere JDBC driver or the jConnect JDBC driver. Passing arguments to `DriverManager.getConnection` establishes the connection. The database environment is an external application from the perspective of the client application.



- **Server-side** When using JDBC within the database server, a connection already exists. The string "jdbc:default:connection" is passed to `DriverManager.getConnection`, which allows the JDBC application to work within the current user connection. This is a quick, efficient, and safe operation because the client application has already passed the database security to establish the connection. The user ID and password, having been provided once, do not need to be provided again. The server-side JDBC driver can only connect to the database of the current connection.

You can write JDBC classes so that they can run both at the client and at the server by employing a single conditional statement for constructing the URL. An external connection requires the host name and port number, while the internal connection requires "jdbc:default:connection".

## SQL Anywhere JDBC drivers

The SQL Anywhere JDBC 3.0 and 4.0 drivers provide some performance benefits and feature benefits compared to the pure Java `jConnect` JDBC driver, however, these drivers do not provide a pure-Java solution. The SQL Anywhere JDBC 4.0 driver is recommended.

### See also

- [“JDBC drivers” on page 404](#)

## Loading the SQL Anywhere JDBC 3.0 driver

Ensure that the SQL Anywhere JDBC 3.0 driver is in your class file path.

```
set classpath=%SQLANY12%\java\sajdbc.jar;%classpath%
```

Before you can use the SQL Anywhere JDBC 3.0 driver in your application, you must load the appropriate driver. Load the SQL Anywhere JDBC 3.0 driver with the following statement:

```
DriverManager.registerDriver( (Driver)
    Class.forName(
        "sybase.jdbc.sqlanywhere.IDriver").newInstance()
    );
```

Using the `newInstance` method works around issues in some browsers.

- As the classes are loaded using `Class.forName`, the package containing the SQL Anywhere JDBC driver does not have to be imported using `import` statements.
- `sajdbc.jar` must be in your class file path when you run the application.

```
set classpath=%classpath%;%SQLANY12%\java\sajdbc.jar
```

### Required files

The Java component of the SQL Anywhere JDBC 3.0 driver is included in the `sajdbc.jar` file installed into the `Java` subdirectory of your SQL Anywhere installation. For Windows, the native component is

*dbjdbc12.dll* in the *bin32* or *bin64* subdirectory of your SQL Anywhere installation; for Unix, the native component is *libdbjdbc12.so*. This component must be in the system path.

## Loading the SQL Anywhere JDBC 4.0 driver

Ensure that the SQL Anywhere JDBC 4.0 driver is in your class file path.

```
set classpath=%SQLANY12%\java\sajdbc4.jar;%classpath%
```

The JDBC 4.0 driver takes advantage of the new automatic JDBC driver registration. The driver is automatically loaded at execution startup when it is in the class file path.

### Required files

The Java component of the SQL Anywhere JDBC 4.0 driver is included in the *sajdbc4.jar* file installed into the *Java* subdirectory of your SQL Anywhere installation. For Windows, the native component is *dbjdbc12.dll* in the *bin32* or *bin64* subdirectory of your SQL Anywhere installation; for Unix, the native component is *libdbjdbc12.so*. This component must be in the system path.

## Supplying a URL to the driver

To connect to a database via a SQL Anywhere JDBC driver, you need to supply a URL for the database. For example:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:DSN=SQL Anywhere 12 Demo" );
```

The URL contains **jdbc:sqlanywhere:** followed by a connection string. If the *sajdbc4.jar* file is in your class file path, then the JDBC 4.0 driver will have been loaded by default and it will handle the URL. Otherwise, the JDBC 3.0 driver will respond to the same URL, provided that it is in the class file path and has been successfully loaded. As shown in the example, an ODBC data source (**DSN**) may be specified for convenience, but you can also use explicit connection parameters, separated by semicolons, in addition to or instead of the data source connection parameter.

If you do not use a data source, you must specify all required connection parameters in the connection string:

```
Connection con = DriverManager.getConnection(
    "jdbc:sqlanywhere:UserID=DBA;Password=sql;Start=..." );
```

The **Driver** connection parameter is not required since neither the ODBC driver nor ODBC driver manager is used. If present, it will be ignored.

### See also

- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## The jConnect JDBC driver

The jConnect driver is available as a separate download at <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>. Documentation for jConnect can also be found on the same page.

If you want to use JDBC from an applet, you must use the jConnect JDBC driver to connect to SQL Anywhere databases.

### The jConnect driver files

jConnect is supplied as a JAR file named *jconn4.jar*. This file is located in your jConnect install location.

### Setting the class file path for jConnect

For your application to use jConnect, the jConnect classes must be in your class file path at compile time and run time, so that the Java compiler and Java runtime can locate the necessary files.

The following command adds the jConnect driver to an existing CLASSPATH environment variable (where *jconnect-path* is your jConnect installation directory).

```
set classpath=jconnect-path\classes\jconn4.jar;%classpath%
```

### Importing the jConnect classes

The classes in jConnect are all in `com.sybase.jdbc4.jdbc`. You must import these classes at the beginning of each source file:

```
import com.sybase.jdbc4.jdbc.*
```

### Encrypting passwords

SQL Anywhere supports password encryption for jConnect connections.

## Installing jConnect system objects into a database

If you want to use jConnect to access system table information (database metadata), you must add the jConnect system objects to your database.

You can add the jConnect system objects to the database when creating the database or at a later time by upgrading the database. You can upgrade a database from Sybase Central or by using the `dbupgrad` utility.

**Note**

Do not add jConnect system objects to your Windows Mobile database.

**Caution**

You should always back up your database files before upgrading. If you apply the upgrade to the existing files, then these files become unusable if the upgrade fails.

### Add jConnect system objects to a database (Sybase Central)

1. Connect to the database from Sybase Central as a user with DBA authority.
2. Click **Tools » SQL Anywhere 12** and then click **Upgrade Database**.
3. Follow the instructions in the **Upgrade Database Wizard**.

### Add jConnect system objects to a database (dbupgrad)

- At a command prompt, run the following command:

```
dbupgrad -c "connection-string"
```

In this command, *connection-string* is a suitable connection string that enables access to a database and server as a user with DBA authority.

### See also

- “jConnect on Windows Mobile” [[SQL Anywhere Server - Database Administration](#)]
- “Backup and data recovery” [[SQL Anywhere Server - Database Administration](#)]

## Loading the jConnect driver

Ensure that the jConnect driver is in your class file path. The driver file *jconn4.jar* is located in the *classes* subdirectory of your jConnect installation.

```
set classpath=.;c:\jConnect-7_0\classes\jconn4.jar;%classpath%
```

The jConnect driver takes advantage of the new automatic JDBC driver registration. The driver is automatically loaded at execution startup when it is in the class file path.

## Supplying a URL to the driver

To connect to a database via jConnect, you need to supply a URL for the database. For example:

```
Connection con = DriverManager.getConnection(  
    "jdbc:sybase:Tds:localhost:2638", "DBA", "sql");
```

The URL is composed in the following way:

```
jdbc:sybase:Tds:host:port
```

The individual components are:

- **jdbc:sybase:Tds** The jConnect JDBC driver, using the TDS application protocol.
- **host** The IP address or name of the computer on which the server is running. If you are establishing a same-host connection, you can use localhost, which means the computer system you are logged into.

- **port** The port number on which the database server listens. The port number assigned to SQL Anywhere is 2638. Use that number unless there are specific reasons not to do so.

The connection string must be less than 253 characters in length.

If you are using the SQL Anywhere personal server, make sure to include the TCP/IP support option when starting the server.

## Specifying a database on a server

Each SQL Anywhere database server can have one or more databases loaded at a time. If the URL you supply when connecting via jConnect specifies a server, but does not specify a database, then the connection attempt is made to the default database on the server.

You can specify a particular database by providing an extended form of the URL in one of the following ways.

### Using the ServiceName parameter

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

The question mark followed by a series of assignments is a standard way of providing arguments to a URL. The case of ServiceName is not significant, and there must be no spaces around the = sign. The *database* parameter is the database name, not the server name. The database name must not include the path or file suffix. For example:

```
Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA", "sql");
```

### Using the RemotePWD parameter

A workaround exists for passing additional connection parameters to the server.

This technique allows you to provide additional connection parameters such as the database name, or a database file, using the RemotePWD field. You set RemotePWD as a Properties field using the put method.

The following code illustrates how to use the field.

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "sql" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

As shown in the example, a comma must precede the DatabaseFile connection parameter. Using the DatabaseFile parameter, you can start a database on a server using jConnect. By default, the database is

started with AutoStop=YES. If you specify utility\_db with a DatabaseFile (DBF) or DatabaseName (DBN) connection parameter (for example, DBN=utility\_db), then the utility database is started automatically.

### See also

- [“Utility databases” \[SQL Anywhere Server - Database Administration\]](#)

## Database options set for jConnect connections

When an application connects to the database using the jConnect driver, the sp\_tsql\_environment stored procedure is called. The sp\_tsql\_environment procedure sets some database options for compatibility with Adaptive Server Enterprise behavior.

### See also

- [“Characteristics of Sybase Open Client and jConnect connections” \[SQL Anywhere Server - Database Administration\]](#)
- [“sp\\_tsql\\_environment system procedure” \[SQL Anywhere Server - SQL Reference\]](#)

## Connections from a JDBC client application

Database metadata is always available when using a SQL Anywhere JDBC driver.

If you want to access database system tables (database metadata) from a JDBC application that uses jConnect, you must add a set of jConnect system objects to your database. These procedures are installed to all databases by default. The dbinit -i option prevents this installation.

For more information about adding the jConnect system objects to a database, see [“The jConnect JDBC driver” on page 409](#).

The following complete Java application is a command line program that connects to a running database, prints a set of information to your command line, and terminates.

Establishing a connection is the first step any JDBC application must take when working with database data.

This example illustrates an external connection, which is a regular client/server connection. For information about how to create an internal connection from Java classes running inside the database server, see [“Establishing a connection from a server-side JDBC class” on page 415](#).

### Connection example code

The following example uses the JDBC 4.0 version of the SQL Anywhere JDBC driver by default to connect to the database. To use a different driver, you can pass in the driver name (jdbc4, jdbc3, jConnect) on the command line. Examples for using the JDBC 4.0 driver, the JDBC 3.0 driver, and jConnect are included in the code. This example assumes that a database server has already been started

using the sample database. The source code can be found in the file *JDBCConnect.java* in the *%SQLANYSAMP12%\SQLAnywhere\JDBC* directory.

```
import java.io.*;
import java.sql.*;

public class JDBCConnect
{
    public static void main( String args[] )
    {
        try
        {
            String arg;
            Connection con;

            // Select the JDBC driver and create a connection.
            // May throw a SQLException.
            // Choices are:
            // 1. jConnect driver
            // 2. SQL Anywhere JDBC 3.0 driver
            // 3. SQL Anywhere JDBC 4.0 driver
            arg = "jdbc4";
            if( args.length > 0 ) arg = args[0];
            if( arg.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", "DBA", "sql");
            }
            else if( arg.compareToIgnoreCase( "jdbc3" ) == 0 )
            {
                DriverManager.registerDriver( (Driver)
                    Class.forName(
                        "sybase.jdbc.sqlanywhere.IDriver").newInstance()
                    );
                con = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
            }
            else
            {
                con = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
            }

            System.out.println("Using "+arg+" driver");

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();

            // Create a result set object by executing the query.
            // May throw a SQLException.
            ResultSet rs = stmt.executeQuery(
                "SELECT ID, GivenName, Surname FROM Customers");

            // Process the result set.
            while (rs.next())
            {
                int value = rs.getInt(1);
                String FirstName = rs.getString(2);
                String LastName = rs.getString(3);
                System.out.println(value+" "+FirstName+" "+LastName);
            }
            rs.close();
        }
    }
}
```

```
        stmt.close();
        con.close();
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());

        System.exit(1);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.exit(1);
    }
    System.exit(0);
}
}
```

## How the connection example works

The external connection example is a Java command line program.

### Importing packages

The application requires a couple of packages, which are imported in the first lines of *JDBCConnect.java*:

- The java.io package contains the Java input/output classes, which are required for printing to the command prompt window.
- The java.sql package contains the JDBC classes, which are required for all JDBC applications.

### The main method

Each Java application requires a class with a method named main, which is the method invoked when the program starts. In this simple example, *JDBCConnect.main* is the only public method in the application.

The *JDBCConnect.main* method carries out the following tasks:

1. Determines which driver to load based on the command line argument. For the SQL Anywhere JDBC 3.0 driver, it loads the requested driver (using the *Class.forName* method). The *jConnect 7.0* and the SQL Anywhere JDBC 4.0 driver are automatically loaded at startup if it is in the class file path.
2. Connects to the default running database using the selected JDBC driver URL. The *getConnection* method establishes a connection using the specified URL.
3. Creates a statement object, which is the container for the SQL statement.
4. Creates a result set object by executing a SQL query.
5. Iterates through the result set, printing the column information.
6. Closes each of the result set, statement, and connection objects.



## Running the connection example

### Create and execute the external connection example application

1. At a command prompt, change to the `%SQLANY12%\SQLAnywhere\JDBC` directory.
2. Start a database server with the sample database on your local computer using the following command:

```
dbsrv12 "%SQLANY12%\demo.db"
```

3. Set the CLASSPATH environment variable. The SQL Anywhere JDBC 4.0 driver contained in `sajdbc4.jar` is used in this example.

```
set classpath=.;%SQLANY12%\java\sajdbc4.jar
```

If you are using the SQL Anywhere JDBC 3.0 driver instead, then use the following:

```
set classpath=.;%SQLANY12%\java\sajdbc.jar
```

If you are using the jConnect driver instead, then use the following (where *path* is your jConnect installation directory):

```
set classpath=.;jconnect-path\classes\jconn4.jar
```

4. Run the following command to compile the example:

```
javac JDBCConnect.java
```

5. Run the following command to execute the example:

```
java JDBCConnect
```

Add a command line argument such as `jconnect` or `jdbc3` to load a different JDBC driver.

```
java JDBCConnect jconnect
```

6. Confirm that a list of identification numbers with customer's names appears at a command prompt.

If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required. Check that your class file path is correct. An incorrect setting may result in a failure to locate a class.

## Establishing a connection from a server-side JDBC class

SQL statements in JDBC are built using the `createStatement` method of a `Connection` object. Even classes running inside the server need to establish a connection to create a `Connection` object.

Establishing a connection from a server-side JDBC class is more straightforward than establishing an external connection. Because the user is already connected to the database, the class simply uses the current connection.

## Server-side connection example code

The following is the source code for the server-side connection example. It is a modified version of the *JDBCCConnect.java* example and is located in *%SQLANYXSAMP12%\SQLAnywhere\JDBC\JDBCCConnect2.java*.

```
import java.io.*;
import java.sql.*;

public class JDBCCConnect2
{
    public static void main( String args[] )
    {
        try
        {
            // Open the connection. May throw a SQLException.
            Connection con = DriverManager.getConnection(
                "jdbc:default:connection" );

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();
            // Create a result set object by executing the query.
            // May throw a SQLException.
            ResultSet rs = stmt.executeQuery(
                "SELECT ID, GivenName, Surname FROM Customers");

            // Process the result set.
            while (rs.next())
            {
                int value = rs.getInt(1);
                String FirstName = rs.getString(2);
                String LastName = rs.getString(3);
                System.out.println(value+" "+FirstName+" "+LastName);
            }
            rs.close();
            stmt.close();
            con.close();
        }
        catch (SQLException sqe)
        {
            System.out.println("Unexpected exception : " +
                sqe.toString() + ", sqlstate = " +
                sqe.getSQLState());
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

## How the server-side connection example differs

The server-side connection example is almost identical to the client-side connection example, with the following exceptions:

1. The JDBC driver does not need to be pre-loaded.

2. It connects to the default running database using the current connection. The URL in the `getConnection` call has been changed as follows:

```
Connection con = DriverManager.getConnection(
    "jdbc:default:connection" );
```

3. The `System.exit()` statements have been removed.

## Running the server-side connection example

### Create and execute the internal connection example application

1. At a command prompt, change to the `%SQLANYAMP12%\SQLAnywhere\JDBC` directory.
2. Start a database server with the sample database on your local computer using the following command:

```
dbeng12 "%SQLANYAMP12%\demo.db"
```

3. For server-side JDBC, it is not necessary to set the `CLASSPATH` environment variable.
4. Enter the following command to compile the example:

```
javac JDBConnect2.java
```

5. Install the class into the sample database using Interactive SQL. Execute the following statement:

```
INSTALL JAVA NEW
FROM FILE 'JDBConnect2.class';
```

You can also install the class using Sybase Central. While connected to the sample database, open the **Java** subfolder under **External Environments** and click **File » New » Java Class**. Then follow the instructions in the wizard.

6. Define a stored procedure named `JDBConnect` that acts as a wrapper for the `JDBConnect2.main` method in the class:

```
CREATE PROCEDURE JDBConnect(OUT args LONG VARCHAR)
EXTERNAL NAME 'JDBConnect2.main([Ljava/lang/String;)]V'
LANGUAGE JAVA;
```

7. Call the `JDBConnect2.main` method as follows:

```
CALL JDBConnect();
```

The first time a Java class is called in a session, the Java VM must be loaded. This might take a few seconds.

8. Confirm that a list of identification numbers with customers' names appears in the database server messages window.

If the attempt to connect fails, an error message appears instead. Confirm that you have executed all the steps as required.

## Notes on JDBC connections

- **Autocommit behavior** The JDBC specification requires that, by default, a COMMIT is performed after each data manipulation statement. Currently, the client-side JDBC behavior is to commit (autocommit is true) and the server-side behavior is to not commit (autocommit is false). To obtain the same behavior in both client-side and server-side applications, you can use a statement such as the following:

```
con.setAutoCommit( false );
```

In this statement, con is the current connection object. You could also set autocommit to true.

- **Setting transaction isolation level** To set the transaction isolation level, the application must call the Connection.setTransactionIsolation method with one of the following values.

For the SQL Anywhere JDBC 4.0 driver use:

- sybase.jdbc4.sqlanywhere.IConnection.SA\_TRANSACTION\_SNAPSHOT
- sybase.jdbc4.sqlanywhere.IConnection.SA\_TRANSACTION\_STATEMENT\_SNAPSHOT
- sybase.jdbc4.sqlanywhere.IConnection.SA\_TRANSACTION\_STATEMENT\_READONLY\_SNAPSHOT

For the SQL Anywhere JDBC 3.0 driver use:

- sybase.jdbc.sqlanywhere.IConnection.SA\_TRANSACTION\_SNAPSHOT
- sybase.jdbc.sqlanywhere.IConnection.SA\_TRANSACTION\_STATEMENT\_SNAPSHOT
- sybase.jdbc.sqlanywhere.IConnection.SA\_TRANSACTION\_STATEMENT\_READONLY\_SNAPSHOT

The following example sets the transaction isolation level to SNAPSHOT using the JDBC 4.0 driver.

```
try
{
    con.setTransactionIsolation(
        sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
    );
}
catch( Exception e )
{
    System.err.println( "Error! Could not set isolation level" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
}
```

For more information about the getTransactionIsolation and setTransactionIsolation, see documentation on the java.sql.Connection interface at <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

- **Connection defaults** From server-side JDBC, only the first call to getConnection( "jdbc:default:connection" ) creates a new connection with the default values. Subsequent calls return a wrapper of the current connection with all connection properties unchanged. If you set autocommit to false in your initial connection, any subsequent getConnection calls within the same Java code return a connection with autocommit set to false.

You may want to ensure that closing a connection restores the connection properties to their default values, so that subsequent connections are obtained with standard JDBC values. The following code achieves this:

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");

boolean oldAutoCommit = con.getAutoCommit();
try
{
    // main body of code here
}
finally
{
    con.setAutoCommit( oldAutoCommit );
}
```

This discussion applies not only to autocommit, but also to other connection properties such as transaction isolation level and read-only mode.

For more information about the `getTransactionIsolation`, `setTransactionIsolation`, and `isReadOnly` methods, see documentation on the `java.sql.Connection` interface at <http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/>.

## Data access using JDBC

Java applications that hold some or all classes in the database have significant advantages over traditional SQL stored procedures. At an introductory level, however, it may be helpful to use the parallels with SQL stored procedures to demonstrate the capabilities of JDBC. In the following examples, you write Java classes that insert a row into the Departments table.

As with other interfaces, SQL statements in JDBC can be either **static** or **dynamic**. Static SQL statements are constructed in the Java application and sent to the database. The database server parses the statement, selects an execution plan, and executes the statement. Together, parsing and selecting an execution plan are referred to as **preparing** the statement.

If a similar statement has to be executed many times (many inserts into one table, for example), there can be significant overhead in static SQL because the preparation step has to be executed each time.

In contrast, a dynamic SQL statement contains placeholders. The statement, prepared once using these placeholders, can be executed many times without the additional expense of preparing. Dynamic SQL is discussed in “[Using prepared statements for more efficient access](#)” on page 422.

## Preparing for the examples

### Sample code

The code fragments in this section are taken from the complete class in `%SQLANYXSAMP12%\SQLAnywhere\JDBC\JDBCExample.java`.

### Install the JDBCExample class

1. Compile the *JDBCExample.java* source code.
2. Using Interactive SQL, connect to the sample database as the DBA.
3. Install the *JDBCExample.class* file into the sample database by executing the following statement in Interactive SQL:

```
INSTALL JAVA NEW  
FROM FILE 'JDBCExample.class'
```

If the database server was not started from the same directory as the class file, then you will have to include the path to the class file in the `INSTALL` statement.

You can also install the class using Sybase Central. While connected to the sample database, open the **Java** subfolder under **External Environments** and click **File » New » Java Class**. Follow the instructions in the wizard.

## Inserts, updates, and deletes using JDBC

Static SQL statements such as `INSERT`, `UPDATE`, and `DELETE`, which do not return result sets, are executed using the `executeUpdate` method of the `Statement` class. Statements, such as `CREATE TABLE` and other data definition statements, can also be executed using `executeUpdate`.

The `addBatch`, `clearBatch`, and `executeBatch` methods of the `Statement` class may also be used. Due to the fact that the JDBC specification is unclear on the behavior of the `executeBatch` method of the `Statement` class, the following notes should be considered when using this method with the SQL Anywhere JDBC drivers:

- Processing of the batch stops immediately upon encountering a SQL exception or result set. If processing of the batch stops, then a `BatchUpdateException` will be thrown by the `executeBatch` method. Calling the `getUpdateCounts` method on the `BatchUpdateException` will return an integer array of row counts where the set of counts prior to the batch failure will contain a valid non-negative update count; while all counts at the point of the batch failure and beyond will contain a -1 value. Casting the `BatchUpdateException` to a `SQLException` will provide additional details as to why batch processing was stopped.
- The batch is only cleared when the `clearBatch` method is explicitly called. As a result, calling the `executeBatch` method repeatedly will re-execute the batch over and over again. In addition, calling `execute(sql_query)` or `executeQuery(sql_query)` will correctly execute the specified SQL query, but will not clear the underlying batch. Hence, calling the `executeBatch` method followed by `execute(sql_query)` followed by the `executeBatch` method again will execute the set of batched statements, then execute the specified SQL query, and then execute the set of batched statements again.

The following code fragment illustrates how to execute an `INSERT` statement. It uses a `Statement` object that has been passed to the `InsertStatic` method as an argument.

```

public static void InsertStatic( Statement stmt )
{
    try
    {
        int iRows = stmt.executeUpdate(
            "INSERT INTO Departments (DepartmentID, DepartmentName)"
            + " VALUES (201, 'Eastern Sales'" );
        // Print the number of rows inserted
        System.out.println(iRows + " rows inserted");
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

## Notes

- This code fragment is part of the *JDBCExample.java* file included in the *%SQLANYXSAMP12%\SQLAnywhere\JDBC* directory.
- The `executeUpdate` method returns an integer that reflects the number of rows affected by the operation. In this case, a successful INSERT would return a value of one (1).
- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

## Run the JDBC Insert example

### Run the JDBC Insert example

1. Using Interactive SQL, connect to the sample database as the DBA.
2. Ensure the JDBCExample class has been installed.

For more information about installing the Java examples classes, see [“Preparing for the examples” on page 419](#).

3. Define a stored procedure named JDBCExample that acts as a wrapper for the JDBCExample.main method in the class:

```

CREATE PROCEDURE JDBCExample(IN arg CHAR(50))
EXTERNAL NAME 'JDBCExample.main([Ljava/lang/String;])V'
LANGUAGE JAVA;

```

4. Call the JDBCExample.main method as follows:

```
CALL JDBCExample( 'insert' );
```

The argument string 'insert' causes the InsertStatic method to be invoked.

5. Confirm that a row has been added to the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

6. There is a similar method in the example class called DeleteStatic that shows how to delete the row that has just been added. Call the JDBCExample.main method as follows:

```
CALL JDBCExample( 'delete' );
```

The argument string 'delete' causes the DeleteStatic method to be invoked.

7. Confirm that the row has been deleted from the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

## Using prepared statements for more efficient access

If you use the Statement interface, you parse each statement that you send to the database, generate an access plan, and execute the statement. The steps before execution are called **preparing** the statement.

You can achieve performance benefits if you use the PreparedStatement interface. This allows you to prepare a statement using placeholders, and then assign values to the placeholders when executing the statement.

Using prepared statements is particularly useful when carrying out many similar actions, such as inserting many rows.

### Example

The following example illustrates how to use the PreparedStatement interface, although inserting a single row is not a good use of prepared statements.

The following InsertDynamic method of the JDBCExample class carries out a prepared statement:

```
public static void InsertDynamic( Connection con,
                                String ID, String name )
{
    try
    {
        // Build the INSERT statement
        // ? is a placeholder character
        String sqlStr = "INSERT INTO Departments " +
            "( DepartmentID, DepartmentName ) " +
            "VALUES ( ? , ? )";

        // Prepare the statement
        PreparedStatement stmt =
```



```

        con.prepareStatement( sqlStr );

        // Set some values
        int idValue = Integer.valueOf( ID );
        stmt.setInt( 1, idValue );
        stmt.setString( 2, name );

        // Execute the statement
        int iRows = stmt.executeUpdate();

        // Print the number of rows inserted
        System.out.println(iRows + " rows inserted");
    }
    catch (SQLException sqe)
    {
        System.out.println("Unexpected exception : " +
            sqe.toString() + ", sqlstate = " +
            sqe.getSQLState());
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

## Notes

- This code fragment is part of the *JDBCExample.java* file included in the *%SQLANYXSAMP12%\SQLAnywhere\JDBC* directory.
- The `executeUpdate` method returns an integer that reflects the number of rows affected by the operation. In this case, a successful INSERT would return a value of one (1).
- When run as a server-side class, the output from `System.out.println` goes to the database server messages window.

## Run the JDBC Insert example

### Run the JDBC Insert example

1. Using Interactive SQL, connect to the sample database as the DBA.
2. Ensure the *JDBCExample* class has been installed.

For more information about installing the Java examples classes, see [“Preparing for the examples” on page 419](#).

3. Define a stored procedure named *JDBCInsert* that acts as a wrapper for the *JDBCExample.Insert* method in the class:

```

CREATE PROCEDURE JDBCInsert(IN arg1 INTEGER, IN arg2 CHAR(50))
    EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)V'
LANGUAGE JAVA;

```

4. Call the *JDBCExample.Insert* method as follows:

```

CALL JDBCInsert( 202, 'Southeastern Sales' );

```

The Insert method causes the InsertDynamic method to be invoked.

5. Confirm that a row has been added to the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

6. There is a similar method in the example class called DeleteDynamic that shows how to delete the row that has just been added.

Define a stored procedure named JDBCDelete that acts as a wrapper for the JDBCExample.Delete method in the class:

```
CREATE PROCEDURE JDBCDelete(IN arg1 INTEGER)
  EXTERNAL NAME 'JDBCExample.Delete(I)V'
  LANGUAGE JAVA;
```

7. Call the JDBCExample.Delete method as follows:

```
CALL JDBCDelete( 202 );
```

The Delete method causes the DeleteDynamic method to be invoked.

8. Confirm that the row has been deleted from the Departments table.

```
SELECT * FROM Departments;
```

The example program displays the updated contents of the Departments table in the database server messages window.

### See also

- [“Prepared statements” on page 2](#)

## JDBC batch methods

The addBatch method of the PreparedStatement class is used for performing batched (or wide) inserts. The following are some guidelines to using this method.

1. An INSERT statement should be prepared using one of the prepareStatement methods of the Connection class.

```
// Build the INSERT statement
String sqlStr = "INSERT INTO Departments " +
               "( DepartmentID, DepartmentName ) " +
               "VALUES ( ? , ? )";
// Prepare the statement
PreparedStatement stmt =
  con.prepareStatement( sqlStr );
```

2. The parameters for the prepared insert statement should be set and batched as follows:

```

// loop to batch "n" sets of parameters
for( i=0; i < n; i++ )
{
    // Note "stmt" is the original prepared insert statement from step 1.
    stmt.setSomeType( 1, param_1 );
    stmt.setSomeType( 2, param_2 );
    .
    .
    .
    // Note that there are "m" parameters in the statement.
    stmt.setSomeType( m , param_m );

    // Add the set of parameters to the batch and
    // move to the next row of parameters.
    stmt.addBatch();
}

```

Example:

```

for( i=0; i < 5; i++ )
{
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
    stmt.addBatch();
}

```

3. The batch must be executed using the `executeBatch` method of the `PreparedStatement` class.

Note that BLOB parameters are not supported in batches.

When using the SQL Anywhere JDBC driver to perform batched inserts, it is recommended that you use a small column size. Using batched inserts to insert large binary or character data into long binary or long varchar columns is not recommended and may degrade performance. The performance can decrease because the SQL Anywhere JDBC driver must allocate large amounts of memory to hold each of the batched insert rows. In all other cases, using batched inserts should provide better performance than using individual inserts.

## Returning result sets

This section describes how to make one or more result sets available from Java methods.

You must write a Java method that returns one or more result sets to the calling environment, and wrap this method in a SQL stored procedure. The following code fragment illustrates how multiple result sets can be returned to the caller of this Java procedure. It uses three `executeQuery` statements to obtain three different result sets.

```

public static void Results( ResultSet[] rset )
    throws SQLException
{
    // Demonstrate returning multiple result sets

    Connection con = DriverManager.getConnection(
        "jdbc:default:connection" );
    rset[0] = con.createStatement().executeQuery(
        "SELECT * FROM Employees" +
        "    ORDER BY EmployeeID" );
}

```

```
rset[1] = con.createStatement().executeQuery(
    "SELECT * FROM Departments" +
    " ORDER BY DepartmentID" );
rset[2] = con.createStatement().executeQuery(
    "SELECT i.ID,i.LineID,i.ProductID,i.Quantity," +
    "        s.OrderDate,i.ShipDate," +
    "        s.Region,e.GivenName||' '||e.Surname" +
    " FROM SalesOrderItems AS i" +
    " JOIN SalesOrders AS s" +
    " JOIN Employees AS e" +
    " WHERE s.ID=i.ID" +
    "        AND s.SalesRepresentative=e.EmployeeID" );
con.close();
}
```

## Notes

- This server-side JDBC example is part of the *JDBCExample.java* file included in the `%SQLANYSAMPI2%\SQLAnywhere\JDBC` directory.
- It obtains a connection to the default running database by using `getConnection`.
- The `executeQuery` methods return result sets.

## Run the JDBC result set example

### Run the JDBC result set example

1. Using Interactive SQL, connect to the sample database as the DBA.
2. Ensure the `JDBCExample` class has been installed.

For more information about installing the Java examples classes, see [“Preparing for the examples” on page 419](#).

3. Define a stored procedure named `JDBCResults` that acts as a wrapper for the `JDBCExample.Results` method in the class:

```
CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
    DYNAMIC RESULT SETS 3
    EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)]V'
    LANGUAGE JAVA;
```

4. Set the following Interactive SQL options so you can see all the results of the query:
  - a. Click **Tools » Options**.
  - b. Click **SQL Anywhere**.
  - c. Click the **Results** tab.
  - d. Set the value for **Maximum Number Of Rows To Display** to **5000**.
  - e. Click **Show All Result Sets**.
  - f. Click **OK**.
5. Call the `JDBCExample.Results` method as follows:

```
CALL JDBCResults();
```

6. Check each of the three results tabs, Result Set 1, Result Set 2, and Result Set 3.

## JDBC notes

- **Access permissions** Like all Java classes in the database, classes containing JDBC statements can be accessed by any user if the GRANT EXECUTE statement has granted them permission to execute the stored procedure that is acting as a wrapper for the Java method.
- **Execution permissions** Java classes are executed with the permissions of the connection executing them. This behavior is different from that of stored procedures, which execute with the permissions of the owner.

## JDBC callbacks

The SQL Anywhere JDBC driver supports two asynchronous callbacks, one for handling the SQL MESSAGE statement and the other for validating requests for file transfers. These are similar to the callbacks supported by the SQL Anywhere ODBC driver.

Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.

A message handler routine can be created to intercept these messages. The following is an example of a message handler callback routine.

```
class T_message_handler implements sybase.jdbc4.sqlanywhere.ASAMessageHandler
{
    private final int MSG_INFO      = 0x80 | 0;
    private final int MSG_WARNING   = 0x80 | 1;
    private final int MSG_ACTION    = 0x80 | 2;
    private final int MSG_STATUS    = 0x80 | 3;
    T_message_handler()
    {
    }

    public SQLException messageHandler(SQLException sqe)
    {
        String msg_type = "unknown";

        switch( sqe.getErrorCode() ) {
            case MSG_INFO:      msg_type = "INFO "; break;
            case MSG_WARNING:   msg_type = "WARNING"; break;
            case MSG_ACTION:    msg_type = "ACTION "; break;
            case MSG_STATUS:    msg_type = "STATUS "; break;
        }

        System.out.println( msg_type + ": " + sqe.getMessage() );
        return sqe;
    }
}
```

A client file transfer request can be validated. Before allowing any transfer to take place, the JDBC driver will invoke the validation callback, if it exists. If the client data transfer is being requested during the

execution of indirect statements such as from within a stored procedure, the JDBC driver will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below. The following is an example of a file transfer validation callback routine.

```
class T_filetrans_callback implements
sybase.jdbc4.sqlanywhere.SAValidateFileTransferCallback
{
    T_filetrans_callback()
    {
    }

    public int callback(String filename, int is_write)
    {
        System.out.println( "File transfer granted for file " + filename
+
                            " with an is_write value of " +
is_write );
        return( 1 ); // 0 to disallow, non-zero to allow
    }
}
```

The filename argument is the name of the file to be read or written. The is\_write parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the JDBC driver allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. Note that if the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

The following sample Java application demonstrates the use of the callbacks supported by the SQL Anywhere JDBC 4.0 driver. You need to place the file `%SQLANY12%\java\sajdbc4.jar` in your classpath.

```
import java.io.*;
import java.sql.*;
import java.util.*;

public class callback
{
    public static void main (String args[]) throws IOException
    {
        Connection        con = null;
        Statement          stmt;

        System.out.println ( "Starting... " );
        con = connect();
        if( con == null )
        {
```

```

        return; // exception should already have been reported
    }
    System.out.println ( "Connected... " );
    try
    {
        // create and register message handler callback
        T_message_handler message_worker = new T_message_handler();

        ((sybase.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( message_wor
ker );

        // create and register validate file transfer callback
        T_filetrans_callback filetran_worker = new
T_filetrans_callback();

        ((sybase.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransferCallbac
k( filetran_worker );

        stmt = con.createStatement();

        // execute message statements to force message handler to be
called
        stmt.execute( "MESSAGE 'this is an info    message' TYPE INFO TO
CLIENT" );
        stmt.execute( "MESSAGE 'this is an action message' TYPE ACTION
TO CLIENT" );
        stmt.execute( "MESSAGE 'this is a warning message' TYPE WARNING
TO CLIENT" );
        stmt.execute( "MESSAGE 'this is a status  message' TYPE STATUS
TO CLIENT" );

        System.out.println( "\n=====\\n" );

        stmt.execute( "set temporary option
allow_read_client_file='on'" );
        try
        {
            stmt.execute( "drop procedure read_client_file_test" );
        }
        catch( SQLException dummy )
        {
            // ignore exception if procedure does not exist
        }
        // create procedure that will force file transfer callback to be
called
        stmt.execute( "create procedure read_client_file_test()" +
            "begin" +
            "    declare v long binary;" +
            "    set v = read_client_file('sample.txt');" +
            "end" );

        // call procedure to force validate file transfer callback to be
called
        try
        {
            stmt.execute( "call read_client_file_test()" );
        }
        catch( SQLException filetrans_exception )
        {
            // Note: Since the file transfer callback returns 1,
            // do not expect a SQL exception to be thrown
            System.out.println( "SQLException: " +
                filetrans_exception.getMessage() );
        }
    }

```

```
        stmt.close();
        con.close();
        System.out.println( "Disconnected" );
    }
    catch( SQLException sqe )
    {
        printExceptions(sqe);
    }
}

private static Connection connect()
{
    Connection connection;

    System.out.println( "Using jdbc4 driver" );
    try
    {
        connection = DriverManager.getConnection(
            "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
    }
    catch( Exception e )
    {
        System.err.println( "Error! Could not connect" );
        System.err.println( e.getMessage() );
        printExceptions( (SQLException)e );
        connection = null;
    }
    return connection;
}

static private void printExceptions(SQLException sqe)
{
    while (sqe != null)
    {
        System.out.println("Unexpected exception : " +
            "SqlState: " + sqe.getSQLState() +
            " " + sqe.toString() +
            ", ErrorCode: " + sqe.getErrorCode());
        System.out.println( "=====\n" );
        sqe = sqe.getNextException();
    }
}
}
```

### See also

- [“MESSAGE statement” \[SQL Anywhere Server - SQL Reference\]](#)

## JDBC escape syntax

You can use JDBC escape syntax from any JDBC application, including Interactive SQL. This escape syntax allows you to call stored procedures regardless of the database management system you are using. The general form for the escape syntax is

{ **keyword** *parameters* }

The set of keywords includes the following:



- **{d date-string}** The date string is any date value accepted by SQL Anywhere.
- **{t time-string}** The time string is any time value accepted by SQL Anywhere.
- **{ts date-string time-string}** The date/time string is any timestamp value accepted by SQL Anywhere.
- **{guid uuid-string}** The uuid-string is any valid GUID string, for example, 41dfe9ef-db91-11d2-8c43-006008d26a6f.
- **{oj outer-join-expr}** The outer-join-expr is a valid OUTER JOIN expression accepted by SQL Anywhere.
- **{? = call func(p1,...)}** The function is any valid function call accepted by SQL Anywhere.
- **{call proc(p1,...)}** The procedure is any valid stored procedure call accepted by SQL Anywhere.
- **{fn func(p1,...)}** The function is any one of the library of functions listed below.

You can use the escape syntax to access a library of functions implemented by the JDBC driver that includes number, string, time, date, and system functions.

For example, to obtain the current date in a database management system-neutral way, you would execute the following:

```
SELECT { FN CURDATE() }
```

The functions that are available depend on the JDBC driver that you are using. The following tables list the functions that are supported by the SQL Anywhere JDBC driver and by the jConnect driver.

### SQL Anywhere JDBC driver supported functions

Numeric functions	String functions	System functions	Time/date functions
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH		CURRENT_TIME-STAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK

<b>Numeric functions</b>	<b>String functions</b>	<b>System functions</b>	<b>Time/date functions</b>
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		hour
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

**jConnect supported functions**

<b>Numeric functions</b>	<b>String functions</b>	<b>System functions</b>	<b>Time/date functions</b>
ABS	ASCII	DATABASE	CURDATE
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		hour

Numeric functions	String functions	System functions	Time/date functions
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

A statement using the escape syntax should work in SQL Anywhere, Adaptive Server Enterprise, Oracle, SQL Server, or another database management system to which you are connected.

In Interactive SQL, the braces *must* be doubled. There must not be a space between successive braces: "{{" is acceptable, but "{ {" is not. As well, you cannot use newline characters in the statement. The escape syntax cannot be used in stored procedures because they are not parsed by Interactive SQL.

For example, to obtain database properties with the sa\_db\_info procedure using SQL escape syntax, you would execute the following in Interactive SQL:

```
{{CALL sa_db_info( 0 ) }}
```

## JDBC 3.0/4.0 API support

All mandatory classes and methods of the JDBC 3.0 and 4.0 specifications are supported by the SQL Anywhere JDBC driver. Some optional methods of the `java.sql.Blob` interface are not supported. These optional methods are:

- `long position( Blob pattern, long start );`
- `long position( byte[] pattern, long start );`
- `OutputStream setBinaryStream( long pos )`
- `int setBytes( long pos, byte[] bytes )`
- `int setBytes( long pos, byte[] bytes, int offset, int len );`
- `void truncate( long len );`

---

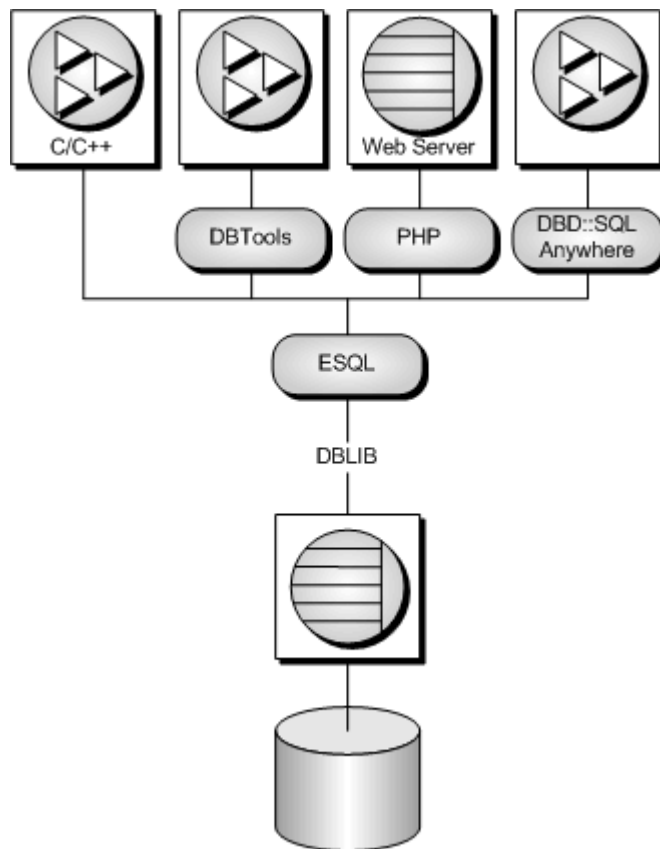
# Embedded SQL

SQL statements embedded in a C or C++ source file are referred to as embedded SQL. A preprocessor translates these statements into calls to a runtime library. Embedded SQL is an ISO/ANSI and IBM standard.

Embedded SQL is portable to other databases and other environments, and is functionally equivalent in all operating environments. It is a comprehensive, low-level interface that provides all the functionality available in the product. Embedded SQL requires knowledge of C or C++ programming languages.

## Embedded SQL applications

You can develop C or C++ applications that access the SQL Anywhere server using the SQL Anywhere embedded SQL interface. The command line database tools are examples of applications developed in this manner.



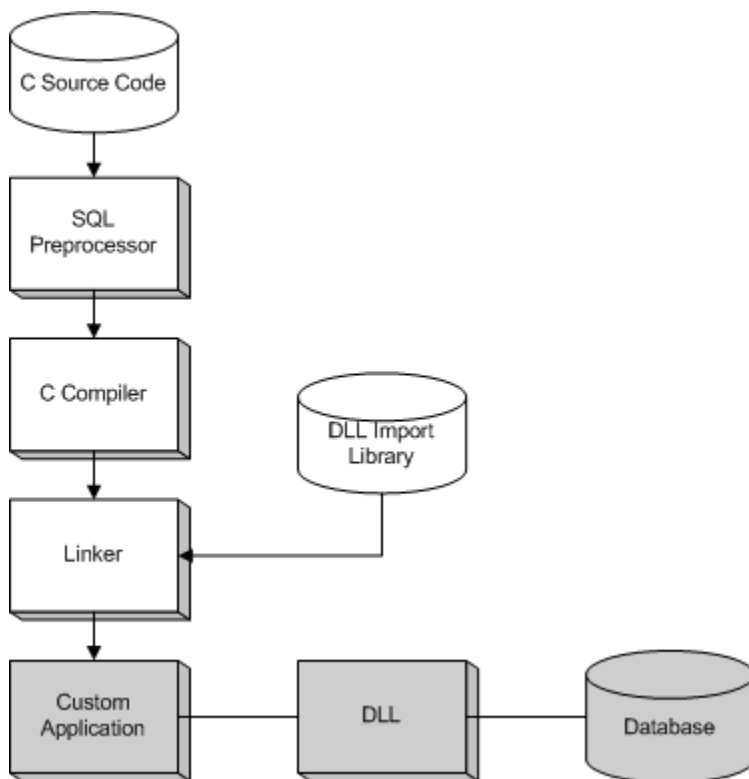
Embedded SQL is a database programming interface for the C and C++ programming languages. It consists of SQL statements intermixed with (embedded in) C or C++ source code. These SQL statements are translated by a **SQL preprocessor** into C or C++ source code, which you then compile.

At runtime, embedded SQL applications use a SQL Anywhere **interface library** called DBLIB to communicate with a database server. DBLIB is a dynamic link library (**DLL**) or shared object on most platforms.

- On Windows operating systems, the interface library is *dblib12.dll*.
- On Unix operating systems, the interface library is *libdblib12.so*, *libdblib12.sl*, or *libdblib12.a*, depending on the operating system.
- On Mac OS X, the interface library is *libdblib12.dylib.1*.

SQL Anywhere provides two flavors of embedded SQL. Static embedded SQL is simpler to use, but is less flexible than dynamic embedded SQL.

## Development process overview



Once the program has been successfully preprocessed and compiled, it is linked with the **import library** for DBLIB to form an executable file. When the database server is running, this executable file uses DBLIB to interact with the database server. The database server does not have to be running when the program is preprocessed.

For Windows, there are 32-bit and 64-bit import libraries for Microsoft Visual C++. The use of import libraries is one method for developing applications that call functions in DLLs. However, it is recommended that the library be dynamically loaded, avoiding the use of import libraries.

### See also

- [“DBLIB dynamic loading under Windows” on page 444](#)

## The SQL preprocessor

The SQL preprocessor is an executable named *sqlpp*.

The preprocessor command line is as follows:

```
sqlpp [ options ] sql-filename [ output-filename ]
```

The preprocessor translates the embedded SQL statements in a C or C++ source file into C code and places the result in an output file. A C or C++ compiler is then used to process the output file. The normal extension for source programs with embedded SQL is *.sqlc*. The default output file name is the *sql-filename* with an extension of *.c*. If the *sql-filename* has a *.c* extension, then the default output file name extension will change to *.cc*.

### Note

When an application is rebuilt to use a new major version of the database interface library, the embedded SQL files must be preprocessed with the same version's SQL preprocessor.

The following table describes the preprocessor options.

Option	Description
<b>-d</b>	Generate code that reduces data space size. Data structures are reused and initialized at execution time before use. This increases code size.

Option	Description
<p><b>-e</b> <i>level</i></p>	<p>Flag as an error any static embedded SQL that is not part of a specified standard. The <i>level</i> value indicates the standard to use. For example, <code>sqlpp -e c03 . . .</code> flags any syntax that is not part of the core SQL/2008 standard. The supported <i>level</i> values are:</p> <ul style="list-style-type: none"> <li>• <b>c08</b> Flag syntax that is not core SQL/2008 syntax</li> <li>• <b>p08</b> Flag syntax that is not full SQL/2008 syntax</li> <li>• <b>c03</b> Flag syntax that is not core SQL/2003 syntax</li> <li>• <b>p03</b> Flag syntax that is not full SQL/2003 syntax</li> <li>• <b>c99</b> Flag syntax that is not core SQL/1999 syntax</li> <li>• <b>p99</b> Flag syntax that is not full SQL/1999 syntax</li> <li>• <b>e92</b> Flag syntax that is not entry-level SQL/1992 syntax</li> <li>• <b>i92</b> Flag syntax that is not intermediate-level SQL/1992 syntax</li> <li>• <b>f92</b> Flag syntax that is not full-SQL/1992 syntax</li> <li>• <b>t</b> Flag non-standard host variable types</li> <li>• <b>u</b> Flag syntax that is not supported by UltraLite</li> </ul> <p>For compatibility with previous SQL Anywhere versions, you can also specify <i>e</i>, <i>i</i>, and <i>f</i>, which correspond to <i>e92</i>, <i>i92</i>, and <i>f92</i>, respectively.</p>
<p><b>-h</b> <i>width</i></p>	<p>Limit the maximum length of lines output by <code>sqlpp</code> to <i>width</i>. The continuation character is a backslash (\) and the minimum value of <i>width</i> is 10.</p>
<p><b>-k</b></p>	<p>Notify the preprocessor that the program to be compiled includes a user declaration of <code>SQLCODE</code>. The definition must be of type <code>long</code>, but does not need to be in a declaration section.</p>
<p><b>-m</b> <i>mode</i></p>	<p>Specify the cursor updatability mode if it is not specified explicitly in the embedded SQL application. The <i>mode</i> can be one of:</p> <ul style="list-style-type: none"> <li>• <b>HISTORICAL</b> In previous versions, embedded SQL cursors defaulted to either <code>FOR UPDATE</code> or <code>READ ONLY</code> (depending on the query and the <code>ansi_update_constraints</code> option value). Explicit cursor updatability was specified on <code>DECLARE CURSOR</code>. Use this option to preserve this behavior.</li> <li>• <b>READONLY</b> Embedded SQL cursors default to <code>READ ONLY</code>. Explicit cursor updatability is specified on <code>PREPARE</code>. This is the default behavior. <code>READ ONLY</code> cursors can result in improved performance.</li> </ul>



Option	Description
<b>-n</b>	Generate line number information in the C file. This consists of <i>#line</i> directives in the appropriate places in the generated C code. If the compiler that you are using supports the <i>#line</i> directive, this option makes the compiler report errors on line numbers in the SQC file (the one with the embedded SQL) as opposed to reporting errors on line numbers in the C file generated by the SQL preprocessor. Also, the <i>#line</i> directives are used indirectly by the source level debugger so that you can debug while viewing the SQC source file.
<b>-o</b> <i>operating-system</i>	Specify the target operating system. The supported operating systems are: <ul style="list-style-type: none"> <li>● <b>WINDOWS</b> Microsoft Windows, including Windows Mobile.</li> <li>● <b>UNIX</b> Use this option if you are creating a 32-bit Unix application.</li> <li>● <b>UNIX64</b> Use this option if you are creating a 64-bit Unix application.</li> </ul>
<b>-q</b>	Quiet mode—do not print messages.
<b>-r-</b>	Generate non-reentrant code.
<b>-s</b> <i>len</i>	Set the maximum size string that the preprocessor puts into the C file. Strings longer than this value are initialized using a list of characters ('a','b','c', and so on). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.
<b>-u</b>	Generate code for UltraLite.

Option	Description
<p><b>-w</b> <i>level</i></p>	<p>Flag as a warning any static embedded SQL that is not part of a specified standard. The <i>level</i> value indicates the standard to use. For example, <code>sqlpp -w c08 . . .</code> flags any SQL syntax that is not part of the core SQL/2008 syntax. The supported <i>level</i> values are:</p> <ul style="list-style-type: none"> <li>• <b>c08</b> Flag syntax that is not core SQL/2008 syntax</li> <li>• <b>p08</b> Flag syntax that is not full SQL/2008 syntax</li> <li>• <b>c03</b> Flag syntax that is not core SQL/2003 syntax</li> <li>• <b>p03</b> Flag syntax that is not full SQL/2003 syntax</li> <li>• <b>c99</b> Flag syntax that is not core SQL/1999 syntax</li> <li>• <b>p99</b> Flag syntax that is not full SQL/1999 syntax</li> <li>• <b>e92</b> Flag syntax that is not entry-level SQL/1992 syntax</li> <li>• <b>i92</b> Flag syntax that is not intermediate-level SQL/1992 syntax</li> <li>• <b>f92</b> Flag syntax that is not full-SQL/1992 syntax</li> <li>• <b>t</b> Flag non-standard host variable types</li> <li>• <b>u</b> Flag syntax that is not supported by UltraLite</li> </ul> <p>For compatibility with previous SQL Anywhere versions, you can also specify e, i, and f, which correspond to e92, i92, and f92, respectively.</p>
<p><b>-x</b></p>	<p>Change multibyte strings to escape sequences so that they can pass through compilers.</p>
<p><b>-z</b> <i>cs</i></p>	<p>Specify the collation sequence. For a list of recommended collation sequences, run <code>dbinit -l</code> at a command prompt.</p> <p>The collation sequence is used to help the preprocessor understand the characters used in the source code of the program, for example, in identifying alphabetic characters suitable for use in identifiers. If <b>-z</b> is not specified, the preprocessor attempts to determine a reasonable collation to use based on the operating system and the SALANG and SACHARSET environment variables.</p>
<p><i>sql-file-name</i></p>	<p>A C or C++ program containing embedded SQL to be processed.</p>
<p><i>output-filename</i></p>	<p>The C language source file created by the SQL preprocessor.</p>

**See also**

- “Embedded SQL” on page 435
- “sql\_flagger\_error\_level option” [*SQL Anywhere Server - Database Administration*]
- “sql\_flagger\_warning\_level option” [*SQL Anywhere Server - Database Administration*]
- “SQLFLAGGER function [Miscellaneous]” [*SQL Anywhere Server - SQL Reference*]
- “sa\_ansi\_standard\_packages system procedure” [*SQL Anywhere Server - SQL Reference*]
- “SQL preprocessor error messages” [*Error Messages*]
- “SQLCA management for multithreaded or reentrant code” on page 463
- “UltraLite Embedded SQL API reference” [*UltraLite - C and C++ Programming*]
- “SACHARSET environment variable” [*SQL Anywhere Server - Database Administration*]
- “SALANG environment variable” [*SQL Anywhere Server - Database Administration*]

## Supported compilers

The C language SQL preprocessor has been used in conjunction with the following compilers:

Operating system	Compiler	Version
Windows	Microsoft Visual C++	6.0 or later
Windows Mobile	Microsoft Visual C++	2005
Unix	GNU or native compiler	

## Embedded SQL header files

All header files are installed in the *SDK\Include* subdirectory of your SQL Anywhere installation directory.

File name	Description
<i>sqlca.h</i>	Main header file included in all embedded SQL programs. This file includes the structure definition for the SQL Communication Area (SQLCA) and prototypes for all embedded SQL database interface functions.
<i>sqlda.h</i>	SQL Descriptor Area structure definition included in embedded SQL programs that use dynamic SQL.
<i>sqldef.h</i>	Definition of embedded SQL interface data types. This file also contains structure definitions and return codes needed for starting the database server from a C program.
<i>sqlerr.h</i>	Definitions for error codes returned in the sqlcode field of the SQLCA.

File name	Description
<i>sqlstate.h</i>	Definitions for ANSI/ISO SQL standard error states returned in the <i>sqlstate</i> field of the <i>SQLCA</i> .
<i>pshpk1.h</i> , <i>pshpk4.h</i> , <i>poppk.h</i>	These headers ensure that structure packing is handled correctly.

## Import libraries

On Windows platforms, all import libraries are installed in the *SDK\Lib* subdirectories, under the SQL Anywhere installation directory. Windows import libraries are stored in the *SDK\Lib\x86* and *SDK\Lib\x64* subdirectories. Windows Mobile import libraries are installed in the *SDK\Lib\CE\Arm.50* subdirectory. An export definition list is stored in *SDK\Lib\Def\dblib.def*.

On Unix platforms, all import libraries are installed in the *lib32* and *lib64* subdirectories, under the SQL Anywhere installation directory.

On Mac OS X platforms, all import libraries are installed in the *System/lib32* and *System/lib64* subdirectories, under the SQL Anywhere installation directory.

Operating system	Compiler	Import library
Windows	Microsoft Visual C++	<i>dblibtm.lib</i>
Windows Mobile	Microsoft Visual C++ 2005	<i>dblib12.lib</i>
Unix (unthreaded applications)	All compilers	<i>libdblib12.so</i> , <i>libdb-tasks12.so</i> , <i>libdblib12.sl</i> , <i>libdbtasks12.sl</i>
Unix (threaded applications)	All compilers	<i>libdblib12_r.so</i> , <i>libdb-tasks12_r.so</i> , <i>libdb-lib12_r.sl</i> , <i>libdb-tasks12_r.sl</i>
Mac OS X (threaded applications)	All compilers	<i>libdblib12.dylib</i> , <i>libdb-tasks12.dylib</i>
Mac OS X (threaded applications)	All compilers	<i>libdblib12_r.dylib</i> , <i>libdb-tasks12_r.dylib</i>

The *libdbtasks12* libraries are called by the *libdblib12* libraries. Some compilers locate *libdbtasks12* automatically. For others, you need to specify it explicitly.

## Sample embedded SQL program

The following is a very simple example of an embedded SQL program.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
{
    db_init( &sqlca );
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    EXEC SQL UPDATE Employees
        SET Surname = 'Plankton'
        WHERE EmployeeID = 195;
    EXEC SQL COMMIT WORK;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 0 );
error:
    printf( "update unsuccessful -- sqlcode = %ld\n",
        sqlca.sqlcode );
    db_fini( &sqlca );
    return( -1 );
}
```

This example connects to the database, updates the last name of employee number 195, commits the change, and exits. There is virtually no interaction between the embedded SQL code and the C code. The only thing the C code is used for in this example is control flow. The WHENEVER statement is used for error checking. The error action (GOTO in this example) is executed after any SQL statement that causes an error.

### See also

- [“Fetching data” on page 479](#)

## Structure of embedded SQL programs

SQL statements are placed (embedded) within regular C or C++ code. All embedded SQL statements start with the words EXEC SQL and end with a semicolon (;). Normal C language comments are allowed in the middle of embedded SQL statements.

Every C program using embedded SQL must contain the following statement before any other embedded SQL statements in the source file.

```
EXEC SQL INCLUDE SQLCA;
```

Every C program using embedded SQL must initialize a SQLCA first:

```
db_init( &sqlca );
```

One of the first embedded SQL statements executed by the C program must be a CONNECT statement. The CONNECT statement is used to establish a connection with the database server and to specify the user ID that is used for authorizing all statements executed during the connection.

Some embedded SQL statements do not generate any C code, or do not involve communication with the database. These statements are allowed before the CONNECT statement. Most notable are the INCLUDE statement and the WHENEVER statement for specifying error processing.

Every C program using embedded SQL must finalize any SQLCA that has been initialized.

```
db_fini( &sqlca );
```

## DBLIB dynamic loading under Windows

The usual practice for developing applications that use functions from DLLs is to link the application against an **import library**, which contains the required function definitions.

This section describes an alternative to using an import library for developing Windows-based embedded SQL applications. DBLIB can be loaded dynamically, without having to link against the import library, using the *esqldll.c* module in the *SDK\C* subdirectory of your installation directory.

A similar technique could be used to dynamically load DBLIB on Unix platforms.

### Load the DBLIB interface DLL dynamically

1. Your program must call `db_init_dll` to load the DLL, and must call `db_fini_dll` to free the DLL. The `db_init_dll` call must be before any function in the database interface, and no function in the interface can be called after `db_fini_dll`.

You must still call the `db_init` and `db_fini` library functions.

2. You must include the *esqldll.h* header file before the EXEC SQL INCLUDE SQLCA statement or include *sqlca.h* in your embedded SQL program. The *esqldll.h* header file includes *sqlca.h*.
3. A SQL OS macro must be defined. The header file *sqlos.h*, which is included by *sqlca.h*, attempts to determine the appropriate macro and define it. However, certain combinations of platforms and compilers may cause this to fail. In this case, you must add a `#define` to the top of this file, or make the definition using a compiler option. The macro that must be defined for Windows is shown below.

Macro	Platforms
<code>_SQL_OS_WINDOWS</code>	All Windows operating systems

4. Compile *esqldll.c*.
5. Instead of linking against the import library, link the object module *esqldll.obj* with your embedded SQL application objects.

### Sample

You can find a sample program illustrating how to load the interface library dynamically in the `%SQLANY%SAMP12%\SQLAnywhere\ESQLDynamicLoad` directory. The source code is in *sample.sqc*.

## Sample embedded SQL programs

Sample embedded SQL programs are included with the SQL Anywhere installation. They are placed in the `%SQLANYAMP12%\SQLAnywhere\C` directory. For Windows Mobile, an additional example is located in the `\SQLAnywhere\CE\esql_sample` directory.

- The static cursor embedded SQL example, *cur.sqc*, demonstrates the use of static SQL statements.
- The dynamic cursor embedded SQL example, *dcur.sqc*, demonstrates the use of dynamic SQL statements.

To reduce the amount of code that is duplicated by the sample programs, the mainlines and the data printing functions have been placed into a separate file. This is *mainch.c* for character mode systems and *mainwin.c* for windowing environments.

The sample programs each supply the following three routines, which are called from the mainlines:

- **WSQLEX\_Init** Connects to the database and opens the cursor.
- **WSQLEX\_Process\_Command** Processes commands from the user, manipulating the cursor as necessary.
- **WSQLEX\_Finish** Closes the cursor and disconnects from the database.

The function of the mainline is to:

1. Call the `WSQLEX_Init` routine.
2. Loop, getting commands from the user and calling `WSQL_Process_Command` until the user quits.
3. Call the `WSQLEX_Finish` routine.

Connecting to the database is done with the embedded SQL `CONNECT` statement supplying the appropriate user ID and password.

In addition to these samples, you may find other programs and source files as part of SQL Anywhere that demonstrate features available for particular platforms.

## Building the sample programs

Files to build the sample programs are supplied with the sample code.

- For Windows, use *build.bat* or *build64.bat* to compile the sample programs.

For x64 platform builds, you may need to set up the correct environment for compiling and linking. Here is an example that builds the sample programs for an x64 platform.

```
set mssdk=c:\MSSDK\v7.0
build64
```

- For Unix, use the shell script *build.sh*.
- For Windows Mobile, use the *esql\_sample.sln* project file for Microsoft Visual C++. This file appears in *\SQLAnywhere\CE\esql\_sample*.

This builds the following examples.

- **CUR** An embedded SQL static cursor example.
- **DCUR** An embedded SQL dynamic cursor example.
- **ODBC** An ODBC example. This is not an embedded SQL example and is described elsewhere.

### See also

- [“ODBC samples” on page 354](#)

## Running the sample programs

The executable files and corresponding source code are located in the *%SQLANYXSAMP12%\SQLAnywhere\C* directory. For Windows Mobile, an additional example is located in the *\SQLAnywhere\CE\esql\_sample* directory.

The Windows versions of the example programs use the Windows graphical user interface. However, to keep the user interface code relatively simple, some simplifications have been made. In particular, these applications do not repaint their Windows on WM\_PAINT messages except to reprint the prompt.

### Run the static cursor sample program

1. Start the SQL Anywhere sample database, *demo.db*.
2. For 32-bit Windows, run the file *curwin.exe*.  
For 64-bit Windows, run the file *curx64.exe*.  
For Unix, run the file *cur*.

3. Follow the on-screen instructions.

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command that you want to perform. Some systems may require you to press Enter after the letter.

### Run the dynamic cursor sample program

1. For 32-bit Windows, run the file *dcurwin.exe*.  
For 64-bit Windows, run the file *dcurx64.exe*.  
For Unix, run the file *dcur*.



2. Each sample program presents a console-type user interface and prompts you for a command. Enter the following connection string to connect to the sample database:

```
DSN=SQL Anywhere 12 Demo
```

3. Each sample program prompts you for a table. Choose one of the tables in the sample database. For example, you can enter **Customers** or **Employees**.
4. Follow the on-screen instructions.

The various commands manipulate a database cursor and print the query results on the screen. Enter the letter of the command you want to perform. Some systems may require you to press Enter after the letter.

## Static cursor sample

This example demonstrates the use of cursors. The particular cursor used here retrieves certain information from the Employees table in the sample database. The cursor is declared statically, meaning that the actual SQL statement to retrieve the information is hard coded into the source program. This is a good starting point for learning how cursors work. The Dynamic Cursor sample takes this first example and converts it to use dynamic SQL statements.

The `open_cursor` routine both declares a cursor for the specific SQL query and also opens the cursor.

Printing a page of information is done by the `print` routine. It loops *pagesize* times, fetching a single row from the cursor and printing it out. The `fetch` routine checks for warning conditions, such as rows that cannot be found (SQLCODE 100), and prints appropriate messages when they arise. In addition, the cursor is repositioned by this program to the row before the one that appears at the top of the current page of data.

The `move`, `top`, and `bottom` routines use the appropriate form of the `FETCH` statement to position the cursor. Note that this form of the `FETCH` statement doesn't actually get the data—it only positions the cursor. Also, a general relative positioning routine, `move`, has been implemented to move in either direction depending on the sign of the parameter.

When the user quits, the cursor is closed and the database connection is also released. The cursor is closed by a `ROLLBACK WORK` statement, and the connection is released by a `DISCONNECT`.

### See also

- [“Sample embedded SQL programs” on page 445](#)
- [“Dynamic cursor sample” on page 447](#)

## Dynamic cursor sample

This sample demonstrates the use of cursors for a dynamic SQL `SELECT` statement. It is a slight modification of the static cursor example. If you have not yet looked at Static Cursor sample, it would be helpful to do so before looking at this sample.

The dcur program allows the user to select a table to look at with the N command. The program then presents as much information from that table as fits on the screen.

When this program is run, it prompts for a connection string. The following is an example.

```
UID=DBA;PWD=sql;DBF=demo.db
```

The C program with the embedded SQL is located in the `%SQLANYSAMPI2%\SQLAnywhere\C` directory. For Windows Mobile, a dynamic cursor example is located in the `\SQLAnywhere\CE\esql_sample` directory. The program looks much like the static cursor sample with the exception of the connect, open\_cursor, and print functions.

The connect function uses the embedded SQL interface function `db_string_connect` to connect to the database. This function provides the extra functionality to support the connection string that is used to connect to the database.

The open\_cursor routine first builds the SELECT statement

```
SELECT * FROM table-name
```

where *table-name* is a parameter passed to the routine. It then prepares a dynamic SQL statement using this string.

The embedded SQL DESCRIBE statement is used to fill in the SQLDA structure with the results of the SELECT statement.

**Note**

An initial guess is taken for the size of the SQLDA (3). If this is not big enough, the actual size of the SELECT list returned by the database server is used to allocate a SQLDA of the correct size.

The SQLDA structure is then filled with buffers to hold strings that represent the results of the query. The fill\_s\_sqlda routine converts all data types in the SQLDA to DT\_STRING and allocates buffers of the appropriate size.

A cursor is then declared and opened for this statement. The rest of the routines for moving and closing the cursor remain the same.

The fetch routine is slightly different: it puts the results into the SQLDA structure instead of into a list of host variables. The print routine has changed significantly to print results from the SQLDA structure up to the width of the screen. The print routine also uses the name fields of the SQLDA to print headings for each column.

**See also**

- [“Sample embedded SQL programs” on page 445](#)
- [“Static cursor sample” on page 447](#)

## Embedded SQL data types

To transfer information between a program and the database server, every piece of data must have a data type. The embedded SQL data type constants are prefixed with `DT_`, and can be found in the `sqldef.h` header file. You can create a host variable of any one of the supported types. You can also use these types in a `SQLDA` structure for passing data to and from the database.

You can define variables of these data types using the `DECL_` macros listed in `sqlca.h`. For example, a variable holding a `BIGINT` value could be declared with `DECL_BIGINT`.

The following data types are supported by the embedded SQL programming interface:

- **DT\_BIT** 8-bit signed integer.
- **DT\_SMALLINT** 16-bit signed integer.
- **DT\_UNSSMALLINT** 16-bit unsigned integer.
- **DT\_TINYINT** 8-bit signed integer.
- **DT\_BIGINT** 64-bit signed integer.
- **DT\_UNSBIGINT** 64-bit unsigned integer.
- **DT\_INT** 32-bit signed integer.
- **DT\_UNSENT** 32-bit unsigned integer.
- **DT\_FLOAT** 4-byte floating-point number.
- **DT\_DOUBLE** 8-byte floating-point number.
- **DT\_DECIMAL** Packed decimal number (proprietary format).
 

```
typedef struct TYPE_DECIMAL {
    char array[1];
} TYPE_DECIMAL;
```
- **DT\_STRING** Null-terminated character string, in the `CHAR` character set. The string is blank-padded if the database is initialized with blank-padded strings.
- **DT\_NSTRING** Null-terminated character string, in the `NCHAR` character set. The string is blank-padded if the database is initialized with blank-padded strings.
- **DT\_DATE** Null-terminated character string that is a valid date.
- **DT\_TIME** Null-terminated character string that is a valid time.
- **DT\_TIMESTAMP** Null-terminated character string that is a valid timestamp.
- **DT\_FIXCHAR** Fixed-length blank-padded character string, in the `CHAR` character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.

- **DT\_NFIXCHAR** Fixed-length blank-padded character string, in the NCHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.
- **DT\_VARCHAR** Varying length character string, in the CHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.

```
typedef struct VARCHAR {
    a_sql_ulen len;
    char      array[1];
} VARCHAR;
```

- **DT\_NVARCHAR** Varying length character string, in the NCHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.

```
typedef struct NVARCHAR {
    a_sql_ulen len;
    char      array[1];
} NVARCHAR;
```

- **DT\_LONGVARCHAR** Long varying length character string, in the CHAR character set.

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
    * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression
    * (may be larger than array_len) */
    char      array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

- **DT\_LONGNVARCHAR** Long varying length character string, in the NCHAR character set. The macro defines a structure, as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
    * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression
    * (may be larger than array_len) */
    char      array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGNVARCHAR structure can be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null-terminated or blank-padded.

- **DT\_BINARY** Varying length binary data with a two-byte length field. The maximum length is 32765 bytes. When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.

```
typedef struct BINARY {
    a_sql_ulen len;
    char      array[1];
} BINARY;
```

- **DT\_LONGBINARY** Long binary data. The macro defines a structure, as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32 array_len; /* number of allocated bytes in array */
    a_sql_uint32 stored_len; /* number of bytes stored in array
                             * (never larger than array_len) */
    a_sql_uint32 untrunc_len; /* number of bytes in untruncated expression
                             * (may be larger than array_len) */
    char        array[1]; /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

The LONGBINARY structure may be used with more than 32767 bytes of data. Large data can be fetched all at once, or in pieces using the GET DATA statement. Large data can be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement.

- **DT\_TIMESTAMP\_STRUCT** SQLDATETIME structure with fields for each part of a timestamp.

```
typedef struct sqldatetime {
    unsigned short year; /* for example 1999 */
    unsigned char  month; /* 0-11 */
    unsigned char  day_of_week; /* 0-6 0=Sunday */
    unsigned short day_of_year; /* 0-365 */
    unsigned char  day; /* 1-31 */
    unsigned char  hour; /* 0-23 */
    unsigned char  minute; /* 0-59 */
    unsigned char  second; /* 0-59 */
    unsigned long  microsecond; /* 0-999999 */
} SQLDATETIME;
```

The SQLDATETIME structure can be used to retrieve fields of DATE, TIME, and TIMESTAMP type (or anything that can be converted to one of these). Often, applications have their own formats and date manipulation code. Fetching data in this structure makes it easier for you to manipulate this data. Note that DATE, TIME, and TIMESTAMP fields can also be fetched and updated with any character type.

If you use a SQLDATETIME structure to enter a date, time, or timestamp into the database, the day\_of\_year and day\_of\_week members are ignored.

- **DT\_VARIABLE** Null-terminated character string. The character string must be the name of a SQL variable whose value is used by the database server. This data type is used only for supplying data to the database server. It cannot be used when fetching data from the database server.

The structures are defined in the *sqlca.h* file. The VARCHAR, NVARCHAR, BINARY, DECIMAL, and LONG data types are not useful for declaring host variables because they contain a one-character array. However, they are useful for allocating variables dynamically or typecasting other variables.

## DATE and TIME database types

There are no corresponding embedded SQL interface data types for the various DATE and TIME database types. These database types are all fetched and updated using either the SQLDATETIME structure or character strings.

### See also

- [“GET DATA statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Sending and retrieving long values” on page 487](#)
- [“date\\_format option” \[SQL Anywhere Server - Database Administration\]](#)
- [“date\\_order option” \[SQL Anywhere Server - Database Administration\]](#)
- [“time\\_format option” \[SQL Anywhere Server - Database Administration\]](#)
- [“timestamp\\_format option” \[SQL Anywhere Server - Database Administration\]](#)

## Host variables in embedded SQL

Host variables are C variables that are identified to the SQL preprocessor. Host variables can be used to send values to the database server or receive values from the database server.

Host variables are quite easy to use, but they have some restrictions. Dynamic SQL is a more general way of passing information to and from the database server using a structure known as the SQL Descriptor Area (SQLDA). The SQL preprocessor automatically generates a SQLDA for each statement in which host variables are used.

Host variables cannot be used in batches. Host variables cannot be used within a subquery in a SET statement.

### See also

- [“Static and dynamic SQL” on page 466](#)

## Host variable declaration

Host variables are defined by putting them into a **declaration section**. According to the ANSI embedded SQL standard, host variables are defined by surrounding the normal C variable declarations with the following:

```
EXEC SQL BEGIN DECLARE SECTION;  
/* C variable declarations */  
EXEC SQL END DECLARE SECTION;
```

These host variables can then be used in place of value constants in any SQL statement. When the database server executes the statement, the value of the host variable is used. Note that host variables cannot be used in place of table or column names: dynamic SQL is required for this. The variable name is prefixed with a colon (:) in a SQL statement to distinguish it from other identifiers allowed in the statement.

In the SQL preprocessor, C language code is only scanned inside a DECLARE SECTION. So, TYPEDEF types and structures are not allowed, but initializers on the variables are allowed inside a DECLARE SECTION.

### Example

The following sample code illustrates the use of host variables on an INSERT statement. The variables are filled in by the program and then inserted into the database:

```
EXEC SQL BEGIN DECLARE SECTION;
long employee_number;
char employee_name[50];
char employee_initials[8];
char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* program fills in variables with appropriate values
*/
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone );
```

### See also

- [“Static cursor sample” on page 447](#)

## C host variable types

Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Macros defined in the *sqlca.h* header file can be used to declare host variables of the following types: NCHAR, VARCHAR, NVARCHAR, LONGVARCHAR, LONGNVARCHAR, BINARY, LONGBINARY, DECIMAL, DT\_FIXCHAR, DT\_NFIXCHAR, DATETIME (SQLDATETIME), BIT, BIGINT, or UNSIGNED BIGINT. They are used as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_NCHAR          v_nchar[10];
DECL_VARCHAR( 10 ) v_varchar;
DECL_NVARCHAR( 10 ) v_nvarchar;
DECL_LONGVARCHAR( 32768 ) v_longvarchar;
DECL_LONGNVARCHAR( 32768 ) v_longnvarchar;
DECL_BINARY( 4000 ) v_binary;
DECL_LONGBINARY( 128000 ) v_longbinary;
DECL_DECIMAL( 30, 6 ) v_decimal;
DECL_FIXCHAR( 10 ) v_fixchar;
DECL_NFIXCHAR( 10 ) v_nfixchar;
DECL_DATETIME      v_datetime;
DECL_BIT           v_bit;
DECL_BIGINT        v_bigint;
DECL_UNSIGNED_BIGINT v_ubigint;
EXEC SQL END DECLARE SECTION;
```

The preprocessor recognizes these macros within an embedded SQL declaration section and treats the variable as the appropriate type. It is recommended that the DECIMAL (DT\_DECIMAL, DECL\_DECIMAL) type not be used since the format of decimal numbers is proprietary.

The following table lists the C variable types that are allowed for host variables and their corresponding embedded SQL interface data types.

C data type	Embedded SQL interface type	Description
<code>short si;</code> <code>short int si;</code>	DT_SMALLINT	16-bit signed integer.
<code>unsigned short int usi;</code>	DT_UNSSMALLINT	16-bit unsigned integer.
<code>long l;</code> <code>long int l;</code>	DT_INT	32-bit signed integer.
<code>unsigned long int ul;</code>	DT_UNSENT	32-bit unsigned integer.
<code>DECL_BIGINT ll;</code>	DT_BIGINT	64-bit signed integer.
<code>DECL_UNSIGNED_BIGINT ull;</code>	DT_UNSBIGINT	64-bit unsigned integer.
<code>float f;</code>	DT_FLOAT	4-byte single-precision floating-point value.
<code>double d;</code>	DT_DOUBLE	8-byte double-precision floating-point value.
<code>char a[n]; /*n&gt;=1*/</code>	DT_STRING	Null-terminated string, in CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.
<code>char *a;</code>	DT_STRING	Null-terminated string, in CHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
<code>DECL_NCHAR a[n]; /*n&gt;=1*/</code>	DT_NSTRING	Null-terminated string, in NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings. This variable holds n-1 bytes plus the null terminator.



C data type	Embedded SQL interface type	Description
DECL_NCHAR *a;	DT_NSTRING	Null-terminated string, in NCHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.
DECL_VARCHAR(n) a;	DT_VARCHAR	Varying length character string, in CHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32765 (bytes).
DECL_NVARCHAR(n) a;	DT_NVARCHAR	Varying length character string, in NCHAR character set, with 2-byte length field. Not null-terminated or blank-padded. The maximum value for n is 32765 (bytes).
DECL_LONGVARCHAR(n) a;	DT_LONGVARCHAR	Varying length long character string, in CHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.
DECL_LONGNVARCHAR(n) a;	DT_LONGNVARCHAR	Varying length long character string, in NCHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.
DECL_BINARY(n) a;	DT_BINARY	Varying length binary data with 2-byte length field. The maximum value for n is 32765 (bytes).
DECL_LONGBINARY(n) a;	DT_LONGBINARY	Varying length long binary data with three 4-byte length fields.

C data type	Embedded SQL interface type	Description
<code>char a; /*n=1*/ DECL_FIXCHAR(n) a;</code>	DT_FIXCHAR	Fixed length character string, in CHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).
<code>DECL_NCHAR a; /*n=1*/ DECL_NFIXCHAR(n) a;</code>	DT_NFIXCHAR	Fixed length character string, in NCHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).
<code>DECL_DATETIME a;</code>	DT_TIMESTAMP_STRUCT	SQLDATETIME structure

### Character sets

For DT\_FIXCHAR, DT\_STRING, DT\_VARCHAR, and DT\_LONGVARCHAR, character data is in the application's CHAR character set, which is usually the character set of the application's locale. An application can change the CHAR character set either by using the CHARSET connection parameter, or by calling the db\_change\_char\_charset function.

For DT\_NFIXCHAR, DT\_NSTRING, DT\_NVARCHAR, and DT\_LONGNVARCHAR, data is in the application's NCHAR character set. By default, the application's NCHAR character set is the same as the CHAR character set. An application can change the NCHAR character set by calling the db\_change\_nchar\_charset function.

### Data lengths

Regardless of the CHAR and NCHAR character sets in use, all data lengths are specified in bytes.

If character set conversion occurs between the server and the application, it is the application's responsibility to ensure that buffers are sufficiently large to handle the converted data, and to issue additional GET DATA statements if data is truncated.

### Pointers to char

The database interface considers a host variable declared as a **pointer to char** (`char * a`) to be 32767 bytes long. Any host variable of type pointer to char used to retrieve information from the database must point to a buffer large enough to hold any value that could possibly come back from the database.

This is potentially quite dangerous because someone could change the definition of the column in the database to be larger than it was when the program was written. This could cause random memory corruption problems. It is better to use a declared array, even as a parameter to a function, where it is passed as a pointer to char. This technique allows the embedded SQL statements to know the size of the array.

## Scope of host variables

A standard host-variable declaration section can appear anywhere that C variables can normally be declared. This includes the parameter declaration section of a C function. The C variables have their normal scope (available within the block in which they are defined). However, since the SQL preprocessor does not scan C code, it does not respect C blocks.

As far as the SQL preprocessor is concerned, host variables are global to the source file; two host variables cannot have the same name.

## See also

- “Locales” [[SQL Anywhere Server - Database Administration](#)]
- “CharSet (CS) connection parameter” [[SQL Anywhere Server - Database Administration](#)]
- “db\_change\_char\_charset function” on page 501
- “db\_change\_nchar\_charset function” on page 502
- “GET DATA statement [ESQL]” [[SQL Anywhere Server - SQL Reference](#)]

## Host variable usage

Host variables can be used in the following circumstances:

- SELECT, INSERT, UPDATE, and DELETE statements in any place where a number or string constant is allowed.
- The INTO clause of SELECT and FETCH statements.
- Host variables can also be used in place of a statement name, a cursor name, or an option name in statements specific to embedded SQL.
- For CONNECT, DISCONNECT, and SET CONNECT statements, a host variable can be used in place of a server name, database name, connection name, user ID, password, or connection string.
- For SET OPTION and GET OPTION, a host variable can be used in place of the option value.

Host variables cannot be used in the following circumstances:

- Host variables cannot be used in place of a table name or a column name in any statement.
- Host variables cannot be used in batches.
- Host variables cannot be used within a subquery in a SET statement.

## SQLCODE and SQLSTATE host variables

The ISO/ANSI standard allows an embedded SQL source file to declare the following special host variables within an embedded SQL declaration section:

```
long SQLCODE;  
char SQLSTATE[6];
```

If used, these variables are set after any embedded SQL statement that makes a database request (EXEC SQL statements other than DECLARE SECTION, INCLUDE, WHENEVER SQLCODE, and so on). As a consequence, the SQLCODE and SQLSTATE host variables must be visible in the scope of every embedded SQL statement that generates database requests.

The following is valid embedded SQL:

```
EXEC SQL INCLUDE SQLCA;
// declare SQLCODE with global scope
EXEC SQL BEGIN DECLARE SECTION;
long SQLCODE;
EXEC SQL END DECLARE SECTION;
sub1() {
    EXEC SQL BEGIN DECLARE SECTION;
    char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;
    exec SQL CREATE TABLE ...
}
sub2() {
    EXEC SQL BEGIN DECLARE SECTION;
    char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;
    exec SQL DROP TABLE ...
}
```

The following is not valid embedded SQL because SQLSTATE is not defined in the scope of the function sub2:

```
EXEC SQL INCLUDE SQLCA;
sub1() {
    EXEC SQL BEGIN DECLARE SECTION;
    char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;
    exec SQL CREATE TABLE...
}
sub2() {
    exec SQL DROP TABLE...
}
```

Note that the SQL preprocessor -k option permits the declaration of the SQLCODE variable outside the scope of an embedded SQL declaration section.

### See also

- [“The SQL preprocessor” on page 437](#)

## Indicator variables

Indicator variables are C variables that hold supplementary information when you are fetching or putting data. There are several distinct uses for indicator variables:

- **NULL values** To enable applications to handle NULL values.
- **String truncation** To enable applications to handle cases when fetched values must be truncated to fit into host variables.

- **Conversion errors** To hold error information.

An indicator variable is a host variable of type `a_sql_len` that is placed immediately following a regular host variable in a SQL statement. For example, in the following INSERT statement, `:ind_phone` is an indicator variable:

```
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

On a fetch or execute where no rows are received from the database server (such as when an error or end of result set occurs), then indicator values are unchanged.

#### Note

To allow for the future use of 32 and 64-bit lengths and indicators, the use of short int for embedded SQL indicator variables is deprecated. Use `a_sql_len` instead.

## Indicator variables: Handling NULL

Do not confuse the SQL concept of NULL with the C-language constant of the same name. In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C-language constant represents a pointer value that does not point to a memory location.

When NULL is used in the SQL Anywhere documentation, it refers to the SQL database meaning given above. The C language constant is referred to as the null pointer (lowercase).

NULL is not the same as any value of the column's defined type. So, something extra is required beyond regular host variables to pass NULL values to the database or receive NULL results back. **Indicator variables** are used for this purpose.

### Using indicator variables when inserting NULL

An INSERT statement could include an indicator variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/*
This program fills in the employee number,
name, initials, and phone number.
*/
if( /* Phone number is unknown */ ) {
ind_phone = -1;
} else {
ind_phone = 0;
}
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

If the indicator variable has a value of -1, a NULL is written. If it has a value of 0, the actual value of `employee_phone` is written.

### Using indicator variables when fetching NULL

Indicator variables are also used when receiving data from the database. They are used to indicate that a NULL value was fetched (indicator is negative). If a NULL value is fetched from the database and an indicator variable is not supplied, an error is generated (SQLE\_NO\_INDICATOR).

## Indicator variables: Using for truncated values

Indicator variables indicate whether any fetched values were truncated to fit into a host variable. This enables applications to handle truncation appropriately.

If a value is truncated on fetching, the indicator variable is set to a positive value, containing the actual length of the database value before truncation. If the length of the value is greater than 32767 bytes, then the indicator variable contains 32767.

## Indicator variables: Handling conversion errors

By default, the `conversion_error` database option is set to On, and any data type conversion failure leads to an error, with no row returned.

You can use indicator variables to tell which column produced a data type conversion failure. If you set the database option `conversion_error` to Off, any data type conversion failure gives a CANNOT\_CONVERT warning, rather than an error. If the column that suffered the conversion error has an indicator variable, that variable is set to a value of -2.

If you set the `conversion_error` option to Off when inserting data into the database, a value of NULL is inserted when a conversion failure occurs.

## Summary of indicator variable values

The following table provides a summary of indicator variable usage.

Indicator value	Supplying value to database	Receiving value from database
> 0	Host variable value	Retrieved value was truncated—actual length in indicator variable.
0	Host variable value	Fetch successful, or <code>conversion_error</code> set to On.
-1	NULL value	NULL result.

Indicator value	Supplying value to database	Receiving value from database
-2	NULL value	Conversion error (when <code>conversion_error</code> is set to Off only). <code>SQLCODE</code> indicates a <code>CANNOT_CONVERT</code> warning.
< -2	NULL value	NULL result.

**See also**

- [“GET DATA statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)

## The SQL Communication Area (SQLCA)

The **SQL Communication Area (SQLCA)** is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application. The SQLCA is used as a handle for the application-to-database communication link. It is passed in to all database library functions that need to communicate with the database server. It is implicitly passed on all embedded SQL statements.

A global SQLCA variable is defined in the interface library. The preprocessor generates an external reference for the global SQLCA variable and an external reference for a pointer to it. The external reference is named `sqlca` and is of type `SQLCA`. The pointer is named `sqlcaptr`. The actual global variable is declared in the import library.

The SQLCA is defined by the `sqlca.h` header file, included in the `SDK\Include` subdirectory of your installation directory.

**SQLCA provides error codes**

You reference the SQLCA to test for a particular error code. The `sqlcode` and `sqlstate` fields contain error codes when a database request has an error. Some C macros are defined for referencing the `sqlcode` field, the `sqlstate` field, and some other fields.

## SQLCA fields

The fields in the SQLCA have the following meanings:

- **sqlcaid** An 8-byte character field that contains the string `SQLCA` as an identification of the SQLCA structure. This field helps in debugging when you are looking at memory contents.
- **sqlcabc** A 32-bit integer that contains the length of the SQLCA structure (136 bytes).
- **sqlcode** A 32-bit integer that specifies the error code when the database detects an error on a request. Definitions for the error codes can be found in the header file `sqlerr.h`. The error code is 0 (zero) for a successful operation, positive for a warning, and negative for an error.

- **sqlerrml** The length of the information in the sqlerrmc field.
- **sqlerrmc** Zero or more character strings to be inserted into an error message. Some error messages contain one or more placeholder strings (%1, %2, ...) that are replaced with the strings in this field.

For example, if a `Table Not Found` error is generated, sqlerrmc contains the table name, which is inserted into the error message at the appropriate place.

- **sqlerrp** Reserved.
- **sqlerrd** A utility array of 32-bit integers.
- **sqlwarn** Reserved.
- **sqlstate** The SQLSTATE status value. The ANSI SQL standard defines this type of return value from a SQL statement in addition to the SQLCODE value. The SQLSTATE value is always a five-character null-terminated string, divided into a two-character class (the first two characters) and a three-character subclass. Each character can be a digit from 0 through 9 or an uppercase alphabetic character A through Z.

Any class or subclass that begins with 0 through 4 or A through H is defined by the SQL standard; other classes and subclasses are implementation defined. The SQLSTATE value '00000' means that there has been no error or warning.

### sqlerror array

The sqlerror field array has the following elements.

- **sqlerrd[1] (SQLIOCOUNT)** The actual number of input/output operations that were required to complete a statement.

The database server does not set this number to zero for each statement. Your program can set this variable to zero before executing a sequence of statements. After the last statement, this number is the total number of input/output operations for the entire statement sequence.

- **sqlerrd[2] (SQLCOUNT)** The value of this field depends on which statement is being executed.
  - **INSERT, UPDATE, PUT, and DELETE statements** The number of rows that were affected by the statement.
  - **OPEN and RESUME statements** On a cursor OPEN or RESUME, this field is filled in with either the actual number of rows in the cursor (a value greater than *or equal to* 0) or an estimate thereof (a negative number whose absolute value is the estimate). It is the actual number of rows if the database server can compute it without counting the rows. The database can also be configured to always return the actual number of rows using the `row_counts` option.
  - **FETCH cursor statement** The SQLCOUNT field is filled if a `SQLE_NOTFOUND` warning is returned. It contains the number of rows by which a `FETCH RELATIVE` or `FETCH ABSOLUTE` statement goes outside the range of possible cursor positions (a cursor can be on a row, before the first row, or after the last row). For a wide fetch, SQLCOUNT is the number of rows actually



fetches, and is less than or equal to the number of rows requested. During a wide fetch, `SQL_NOTFOUND` is only set if no rows are returned.

The value is 0 if the row was not found, but the position is valid, for example, executing `FETCH RELATIVE 1` when positioned on the last row of a cursor. The value is positive if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor.

- **GET DATA statement** The `SQLCOUNT` field holds the actual length of the value.
- **DESCRIBE statement** If the `WITH VARIABLE RESULT` clause is used to describe procedures that may have more than one result set, `SQLCOUNT` is set to one of the following values:
  - **0** The result set may change: the procedure call should be described again following each `OPEN` statement.
  - **1** The result set is fixed. No re-describing is required.

For the `SQL_SYNTAX_ERROR` syntax error, the field contains the approximate character position within the statement where the error was detected.

- **sqlerrd[3] (SQLIOESTIMATE)** The estimated number of input/output operations that are required to complete the statement. This field is given a value on an `OPEN` or `EXPLAIN` statement.

#### See also

- [“SQL Anywhere error messages”](#) [*Error Messages*]
- [“SQL Anywhere error messages sorted by SQLCODE”](#) [*Error Messages*]
- [“SQL Anywhere error messages sorted by SQLSTATE”](#) [*Error Messages*]
- [“Fetching more than one row at a time”](#) on page 483

## SQLCA management for multithreaded or reentrant code

You can use embedded SQL statements in multithreaded or reentrant code. However, if you use a single connection, you are restricted to one active request per connection. In a multithreaded application, you should not use the same connection to the database on each thread unless you use a semaphore to control access.

There are no restrictions on using separate connections on each thread that wants to use the database. The SQLCA is used by the runtime library to distinguish between the different thread contexts. So, each thread wanting to use the database concurrently must have its own SQLCA. The exception is that a thread can use the `db_cancel_request` function to cancel a statement executing on a different thread using that thread's SQLCA.

The following is an example of multithreaded embedded SQL reentrant code.

```
#include <stdio.h>
#include <string.h>
```

```

#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>
#include <windows.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

#define TRUE 1
#define FALSE 0

// multithreading support

typedef struct a_thread_data {
    SQLCA sqlca;
    int num_iters;
    int thread;
    int done;
} a_thread_data;

// each thread's ESQL test

EXEC SQL SET SQLCA "&thread_data->sqlca";

static void PrintSQLError( a_thread_data * thread_data )
/*****
{
    char          buffer[200];

    printf( "%d: SQL error %d -- %s ... aborting\n",
            thread_data->thread,
            SQLCODE,
            sqlerror_message( &thread_data->sqlca,
                             buffer, sizeof( buffer ) ) );
    exit( 1 );
}

EXEC SQL WHENEVER SQLERROR { PrintSQLError( thread_data ); };

static void do_one_iter( void * data )
{
    a_thread_data * thread_data = (a_thread_data *)data;
    int i;
    EXEC SQL BEGIN DECLARE SECTION;
    char user[ 20 ];
    EXEC SQL END DECLARE SECTION;

    if( db_init( &thread_data->sqlca ) != 0 ) {
        for( i = 0; i < thread_data->num_iters; i++ ) {
            EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
            EXEC SQL SELECT USER INTO :user;
            EXEC SQL DISCONNECT;
        }
        printf( "Thread %d did %d iters successfully\n",
                thread_data->thread, thread_data->num_iters );
        db_fini( &thread_data->sqlca );
    }
    thread_data->done = TRUE;
}

int main()
{
    int num_threads = 4;
    int thread;

```

```

int num_iters = 300;
int num_done = 0;
a_thread_data *thread_data;
thread_data = (a_thread_data *)malloc( sizeof(a_thread_data) *
num_threads );
for( thread = 0; thread < num_threads; thread++ ) {
    thread_data[ thread ].num_iters = num_iters;
    thread_data[ thread ].thread = thread;
    thread_data[ thread ].done = FALSE;
    if( _beginthread( do_one_iter,
        8096,
        (void *)&thread_data[thread] ) <= 0 ) {
        printf( "FAILED creating thread.\n" );
        return( 1 );
    }
}
while( num_done != num_threads ) {
    Sleep( 1000 );
    num_done = 0;
    for( thread = 0; thread < num_threads; thread++ ) {
        if( thread_data[ thread ].done == TRUE ) {
            num_done++;
        }
    }
}
return( 0 );
}

```

**See also**

- [“Request management with embedded SQL” on page 494](#)

## Using multiple SQLCAs

You must not use the SQL preprocessor option (-r) that generates non-reentrant code. Reentrant code is a little larger and a little slower because statically initialized global variables cannot be used. However, these effects are minimal.

Each SQLCA used in your program must be initialized with a call to `db_init` and cleaned up at the end with a call to `db_fini`.

The embedded SQL statement `SET SQLCA` is used to tell the SQL preprocessor to use a different SQLCA for database requests. Usually, a statement such as `EXEC SQL SET SQLCA 'task_data->sqlca'`; is used at the top of your program or in a header file to set the SQLCA reference to point at task specific data. Performance is unaffected because this statement does not generate any code. It changes the state within the preprocessor so that any reference to the SQLCA uses the given string.

Each thread must have its own SQLCA. This requirement also applies to code in a shared library (in a DLL, for example) that uses embedded SQL and is called by more than one thread in your application.

You can use the multiple SQLCA support in any of the supported embedded SQL environments, but it is only required in reentrant code.

You do not need to use multiple SQLCAs to connect to more than one database or have more than one connection to a single database.

Each SQLCA can have one unnamed connection. Each SQLCA has an active or current connection.

All operations on a given database connection must use the same SQLCA that was used when the connection was established.

**Note**

Operations on different connections are subject to the normal record locking mechanisms and may cause each other to block and possibly to deadlock.

**See also**

- [“SET SQLCA statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“SET CONNECTION statement \[Interactive SQL\] \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Static and dynamic SQL

There are two ways to embed SQL statements into a C program:

- Static statements
- Dynamic statements

### Static SQL statements

All standard SQL data manipulation and data definition statements can be embedded in a C program by prefixing them with EXEC SQL and suffixing the statement with a semicolon (;). These statements are referred to as **static** statements.

Static statements can contain references to host variables. Host variables can only be used in place of string or numeric constants. They cannot be used to substitute column names or table names; dynamic statements are required to perform those operations.

**See also**

- [“Host variables in embedded SQL” on page 452](#)

### Dynamic SQL statements

In the C language, strings are stored in arrays of characters. Dynamic statements are constructed in C language strings. These statements can then be executed using the PREPARE and EXECUTE statements. These SQL statements cannot reference host variables in the same manner as static statements since the C language variables are not accessible by name when the C program is executing.

To pass information between the statements and the C language variables, a data structure called the **SQL Descriptor Area (SQLDA)** is used. This structure is set up for you by the SQL preprocessor if you specify a list of host variables on the EXECUTE statement in the USING clause. These variables correspond by position to placeholders in the appropriate positions of the prepared statement.

A **placeholder** is put in the statement to indicate where host variables are to be accessed. A placeholder is either a question mark (?) or a host variable reference as in static statements (a host variable name preceded by a colon). In the latter case, the host variable name used in the actual text of the statement serves only as a placeholder indicating a reference to the SQL descriptor area.

A host variable used to pass information to the database is called a **bind variable**.

### Example

```
EXEC SQL BEGIN DECLARE SECTION;
char      comm[200];
char      street[30];
char      city[20];
a_sql_len cityind;
long      empnum;
EXEC SQL END DECLARE SECTION;

...
sprintf( comm,
         "UPDATE %s SET Street = :?, City = :?"
         "WHERE EmployeeID = :?",
         tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
EXEC SQL EXECUTE S1 USING :street, :city:cityind, :empnum;
```

This method requires you to know how many host variables there are in the statement. Usually, this is not the case. So, you can set up your own SQLDA structure and specify this SQLDA in the USING clause on the EXECUTE statement.

The DESCRIBE BIND VARIABLES statement returns the host variable names of the bind variables that are found in a prepared statement. This makes it easier for a C program to manage the host variables. The general method is as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
         "UPDATE %s SET Street = :street, City = :city"
         " WHERE EmployeeID = :empnum",
         tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
/* Assume that there are no more than 10 host variables.
 * See next example if you cannot put a limit on it. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE BIND VARIABLES FOR S1 INTO sqlda;
/* sqlda->sqld will tell you how many
 * host variables there were. */
/* Fill in SQLDA_VARIABLE fields with
 * values based on name fields in sqlda. */
...
EXEC SQL EXECUTE S1 USING DESCRIPTOR sqlda;
free_sqlda( sqlda );
```

## SQLDA contents

The SQLDA consists of an array of variable descriptors. Each descriptor describes the attributes of the corresponding C program variable or the location that the database stores data into or retrieves data from:

- data type
- length if *type* is a string type
- memory address
- indicator variable

## Indicator variables and NULL

The indicator variable is used to pass a NULL value to the database or retrieve a NULL value from the database. The database server also uses the indicator variable to indicate truncation conditions encountered during a database operation. The indicator variable is set to a positive value when not enough space was provided to receive a database value.

## See also

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)
- [“EXECUTE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Indicator variables” on page 458](#)

# Dynamic SELECT statement

A SELECT statement that returns only a single row can be prepared dynamically, followed by an EXECUTE with an INTO clause to retrieve the one-row result. SELECT statements that return multiple rows, however, are managed using dynamic cursors.

With dynamic cursors, results are put into a host variable list or a SQLDA that is specified on the FETCH statement (FETCH INTO and FETCH USING DESCRIPTOR). Since the number of SELECT list items is usually unknown, the SQLDA route is the most common. The DESCRIBE SELECT LIST statement sets up a SQLDA with the types of the SELECT list items. Space is then allocated for the values using the `fill_sqlda` or `fill_s_sqlda` functions, and the information is retrieved by the FETCH USING DESCRIPTOR statement.

The typical scenario is as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
int actual_size;
SQLDA * sqlda;
...
sprintf( comm, "SELECT * FROM %s", table_name );
EXEC SQL PREPARE S1 FROM :comm;
/* Initial guess of 10 columns in result.
   If it is wrong, it is corrected right
   after the first DESCRIBE by reallocating
   sqlda and doing DESCRIBE again. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE SELECT LIST FOR S1
INTO sqlda;
```

```

if( sqlda->sqld > sqlda->sqln )
{
    actual_size = sqlda->sqld;
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( actual_size );
    EXEC SQL DESCRIBE SELECT LIST FOR S1
        INTO sqlda;
}
fill_sqlda( sqlda );
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
EXEC SQL WHENEVER NOTFOUND {break};
for( ;; )
{
    EXEC SQL FETCH C1 USING DESCRIPTOR sqlda;
    /* do something with data */
}
EXEC SQL CLOSE C1;
EXEC SQL DROP STATEMENT S1;

```

**Note**

To avoid consuming unnecessary resources, ensure that statements are dropped after use.

The dynamic cursor example illustrates the use of cursors with a dynamic SELECT statement.

**See also**

- [“PREPARE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“DESCRIBE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Dynamic cursor sample” on page 447](#)
- [“Library function reference” on page 495](#)

## The SQL descriptor area (SQLDA)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file *sqlda.h*.

There are functions in the database interface shared library or DLL that you can use to manage SQLDAs.

When host variables are used with static SQL statements, the preprocessor constructs a SQLDA for those host variables. It is this SQLDA that is actually passed to and from the database server.

**See also**

- [“Library function reference” on page 495](#)

## The SQLDA header file

The contents of *sqlda.h* are as follows:

```

#ifndef _SQLDA_H_INCLUDED
#define _SQLDA_H_INCLUDED

```

```

#define II_SQLDA
#include "sqlca.h"
#if defined( _SQL_PACK_STRUCTURES )
    #if defined( _MSC_VER ) && _MSC_VER > 800
        #pragma warning(push)
        #pragma warning(disable:4103)
    #endif
    #include "pshpk1.h"
#endif
#define SQL_MAX_NAME_LEN    30
#define _sqldafar
typedef short int a_sql_type;

struct sqlname {
    short int    length; /* length of char data */
    char        data[ SQL_MAX_NAME_LEN ]; /* data */
};

struct sqlvar {
    /* array of variable descriptors */
    short int    sqltype; /* type of host variable */
    a_sql_len    sqllen; /* length of host variable */
    void        *sqldata; /* address of variable */
    a_sql_len    *sqlind; /* indicator variable pointer */
    struct sqlname sqlname;
};

#if defined( _SQL_PACK_STRUCTURES )
    #include "poppk.h"
    /* The SQLDA should be 4-byte aligned */
    #include "pshpk4.h"
#endif

struct sqlda {
    unsigned char    sqldaid[8]; /* eye catcher "SQLDA" */
    a_sql_int32      sqldabc; /* length of sqlda structure */
    short int        sqln; /* descriptor size in number of entries */
    short int        sqld; /* number of variables found by DESCRIBE */
    struct sqlvar    sqlvar[1]; /* array of variable descriptors */
};

#define SCALE(sqllen) ((sqllen)/256)
#define PRECISION(sqllen) ((sqllen)&0xff)
#define SET_PRECISION_SCALE(sqllen,precision,scale) \
    sqlllen = (scale)*256 + (precision)
#define DECIMALSTORAGE(sqllen) (PRECISION(sqlllen)/2 + 1)
typedef struct sqlda    SQLDA;
typedef struct sqlvar    SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname    SQLNAME, SQLDA_NAME;
#ifdef SQLDASIZE
#define SQLDASIZE(n) ( sizeof( struct sqlda ) + \
    (n-1) * sizeof( struct sqlvar ) )
#endif
#endif
#include "poppk.h"
#if defined( _MSC_VER ) && _MSC_VER > 800
    #pragma warning(pop)
#endif
#endif

```



## SQLDA fields

The SQLDA fields have the following meanings:

Field	Description
<b>sqldaid</b>	An 8-byte character field that contains the string <b>SQLDA</b> as an identification of the SQLDA structure. This field helps in debugging when you are looking at memory contents.
<b>sqldabc</b>	A 32-bit integer containing the length of the SQLDA structure.
<b>sqln</b>	The number of variable descriptors allocated in the sqlvar array.
<b>sqld</b>	The number of variable descriptors that are valid (contain information describing a host variable). This field is set by the DESCRIBE statement. As well, you can set it when supplying data to the database server.
<b>sqlvar</b>	An array of descriptors of type struct sqlvar, each describing a host variable.

## SQLDA host variable descriptions

Each sqlvar structure in the SQLDA describes a host variable. The fields of the sqlvar structure have the following meanings:

- **sqltype** The type of the variable that is described by this descriptor.

The low order bit indicates whether NULL values are allowed. Valid types and constant definitions can be found in the *sqldef.h* header file.

This field is filled by the DESCRIBE statement. You can set this field to any type when supplying data to the database server or retrieving data from the database server. Any necessary type conversion is done automatically.

- **sqllen** The length of the variable. A sqllen value has type `a_sql_len`. What the length actually means depends on the type information and how the SQLDA is being used.

For LONG VARCHAR, LONG NVARCHAR, and LONG BINARY data types, the `array_len` field of the `DT_LONGVARCHAR`, `DT_LONGNVARCHAR`, or `DT_LONGBINARY` data type structure is used instead of the `sqllen` field.

- **sqldata** A pointer to the memory occupied by this variable. This memory must correspond to the `sqltype` and `sqllen` fields.

For UPDATE and INSERT statements, this variable is not involved in the operation if the `sqldata` pointer is a null pointer. For a FETCH, no data is returned if the `sqldata` pointer is a null pointer. In other words, the column returned by the `sqldata` pointer is an **unbound column**.

If the DESCRIBE statement uses LONG NAMES, this field holds the long name of the result set column. If, in addition, the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type, instead of the column. If the type is a base type, the field is empty.

- **sqlind** A pointer to the indicator value. An indicator value has type `a_sql_len`. A negative indicator value indicates a NULL value. A positive indicator value indicates that this variable has been truncated by a FETCH statement, and the indicator value contains the length of the data before truncation. A value of -2 indicates a conversion error if the `conversion_error` database option is set to Off.

If the `sqlind` pointer is the null pointer, no indicator variable pertains to this host variable.

The `sqlind` field is also used by the DESCRIBE statement to indicate parameter types. If the type is a user-defined data type, this field is set to `DT_HAS_USERTYPE_INFO`. In this case, you may want to perform a DESCRIBE USER TYPES to obtain information on the user-defined data types.

- **sqlname** A VARCHAR-like structure, as follows:

```
struct sqlname {
    short int length;
    char data[ SQL_MAX_NAME_LEN ];
};
```

It is filled by a DESCRIBE statement and is not otherwise used. This field has a different meaning for the two formats of the DESCRIBE statement:

- **SELECT LIST** The name data buffer is filled with the column heading of the corresponding item in the SELECT list.
- **BIND VARIABLES** The name data buffer is filled with the name of the host variable that was used as a bind variable, or "?" if an unnamed parameter marker is used.

On a DESCRIBE SELECT LIST statement, any indicator variables present are filled with a flag indicating whether the SELECT list item is updatable or not. More information about this flag can be found in the `sqldef.h` header file.

If the DESCRIBE statement is a DESCRIBE USER TYPES statement, then this field holds the long name of the user-defined data type instead of the column. If the type is a base type, the field is empty.

### See also

- [“Embedded SQL data types” on page 449](#)
- [“SQLDA sqlen field values” on page 473](#)
- [“conversion\\_error option” \[SQL Anywhere Server - Database Administration\]](#)
- [“Indicator variables” on page 458](#)

## SQLDA sqlen field values

### SQLDA sqlen field values after a DESCRIBE

The DESCRIBE statement gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

The following table indicates the values of the sqlen and sqltype structure members returned by the DESCRIBE statement for the various database types (both SELECT LIST and BIND VARIABLE DESCRIBE statements). For a user-defined database data type, the base type is described.

Your program can use the types and lengths returned from a DESCRIBE, or you may use another type. The database server performs type conversions between any two types. The memory pointed to by the sqldata field must correspond to the sqltype and sqlen fields. The embedded SQL type is obtained by a bitwise AND of sqltype with DT\_TYPES (sqltype & DT\_TYPES).

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR <sup>1</sup>	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
CHAR(n CHAR)	DT_FIXCHAR <sup>1</sup>	n times maximum character length in the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
DATE	DT_DATE	length of longest formatted string

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
DECIMAL(p,s)	DT_DECIMAL	high byte of length field in SQLDA set to p, and low byte set to s
DOUBLE	DT_DOUBLE	8
FLOAT	DT_FLOAT	4
INT	DT_INT	4
LONG BINARY	DT_LONGBINARY	32767
LONG NVARCHAR	DT_LONGVARCHAR / DT_LONGNVARCHAR <sup>2</sup>	32767
LONG VARCHAR	DT_LONGVARCHAR	32767
NCHAR(n)	DT_FIXCHAR / DT_NFIX-CHAR <sup>2</sup>	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGNVARCHAR with a length of 32767 bytes.
NVARCHAR(n)	DT_VARCHAR / DT_NVARCHAR <sup>2</sup>	n times maximum character length in the client's NCHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGNVARCHAR with a length of 32767 bytes.
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2
TIME	DT_TIME	length of longest formatted string
TIMESTAMP	DT_TIMESTAMP	length of longest formatted string
TINYINT	DT_TINYINT	1

Database field type	Embedded SQL type returned	Length (in bytes) returned on describe
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSENT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR <sup>1</sup>	n times maximum data expansion when converting from database character set to the client's CHAR character set. If this length would be more than 32767 bytes, then the embedded SQL type returned is DT_LONGVARCHAR with a length of 32767 bytes.
VARCHAR(n CHAR)	DT_VARCHAR <sup>1</sup>	n times maximum character length in the client's CHAR character set. If this length would be more than 32767, then the embedded SQL type returned is DT_LONGVARCHAR with length 32767.

<sup>1</sup> The type returned for CHAR and VARCHAR may be DT\_LONGVARCHAR if the maximum byte length in the client's CHAR character set is greater than 32767 bytes.

<sup>2</sup> The type returned for NCHAR and NVARCHAR may be DT\_LONGNVARCHAR if the maximum byte length in the client's NCHAR character set is greater than 32767 bytes. NCHAR, NVARCHAR, and LONG NVARCHAR are described by default as either DT\_FIXCHAR, DT\_VARCHAR, or DT\_LONGVARCHAR, respectively. If the db\_change\_nchar\_charset function has been called, the types are described as DT\_NFIXCHAR, DT\_NVARCHAR, and DT\_LONGNVARCHAR, respectively.

### See also

- [“Embedded SQL data types” on page 449](#)
- [“CHAR data type” \[SQL Anywhere Server - SQL Reference\]](#)
- [“NCHAR data type” \[SQL Anywhere Server - SQL Reference\]](#)
- [“NVARCHAR data type” \[SQL Anywhere Server - SQL Reference\]](#)
- [“VARCHAR data type” \[SQL Anywhere Server - SQL Reference\]](#)
- [“db\\_change\\_nchar\\_charset function” on page 502](#)

## SQLDA sqlen field values when sending values

The following table indicates how you specify lengths of values when you supply data to the database server in the SQLDA.

Only the data types displayed in the table are allowed in this case. The DT\_DATE, DT\_TIME, and DT\_TIMESTAMP types are treated the same as DT\_STRING when supplying information to the database; the value must be a null-terminated character string in an appropriate date or time of day format.

Embedded SQL data type	Program action to set the length
DT_BIGINT	No action required.
DT_BINARY(n)	Length taken from field in BINARY structure.
DT_BIT	No action required.
DT_DATE	Length determined by terminating null character.
DT_DOUBLE	No action required.
DT_FIXCHAR(n)	Length field in SQLDA determines length of string.
DT_FLOAT	No action required.
DT_INT	No action required.
DT_LONGBINARY	Length field ignored.
DT_LONGNVARCHAR	Length field ignored.
DT_LONGVARCHAR	Length field ignored.
DT_NFIXCHAR(n)	Length field in SQLDA determines length of string.
DT_NSTRING	Length determined by terminating \0. If the ansi_blanks option is On and the database is blank-padded, then the length field in the SQLDA must be set to the length of the buffer containing the value (at least the length of the value plus space for the terminating null character).
DT_NVARCHAR	Length taken from field in NVARCHAR structure.
DT_SMALLINT	No action required.

Embedded SQL data type	Program action to set the length
DT_STRING	Length determined by terminating \0. If the ansi_blanks option is On and the database is blank-padded, then the length field in the SQLDA must be set to the length of the buffer containing the value (at least the length of the value plus space for the terminating null character).
DT_TIME	Length determined by terminating null character.
DT_TIMESTAMP	Length determined by terminating null character.
DT_TIMESTAMP_STRUCT	No action required.
DT_UNSBIGINT	No action required.
DT_UNSENT	No action required.
DT_UNSSMALLINT	No action required.
DT_VARCHAR(n)	Length taken from field in VARCHAR structure.
DT_VARIABLE	Length determined by terminating \0.

**See also**

- [“Sending LONG data” on page 490](#)

**SQLDA sqlen field values when retrieving data**

The following table indicates the values of the length field when you retrieve data from the database using a SQLDA. The sqlen field is never modified when you retrieve data.

Only the interface data types displayed in the table are allowed in this case. The DT\_DATE, DT\_TIME, and DT\_TIMESTAMP data types are treated the same as DT\_STRING when you retrieve information from the database. The value is formatted as a character string in the current date format.

Embedded SQL data type	What the program must set length field to when receiving	How the database returns length information after fetching a value
DT_BIGINT	No action required.	No action required.
DT_BINARY(n)	Maximum length of BINARY structure (n+2). The maximum value for n is 32765.	len field of BINARY structure set to actual length in bytes.

<b>Embedded SQL data type</b>	<b>What the program must set length field to when receiving</b>	<b>How the database returns length information after fetching a value</b>
DT_BIT	No action required.	No action required.
DT_DATE	Length of buffer.	null character at end of string.
DT_DOUBLE	No action required.	No action required.
DT_FIXCHAR(n)	Length of buffer, in bytes. The maximum value for n is 32767.	Padded with blanks to length of buffer.
DT_FLOAT	No action required.	No action required.
DT_INT	No action required.	No action required.
DT_LONGBINARY	Length field ignored.	Length field ignored.
DT_LONGNVARCHAR	Length field ignored.	Length field ignored.
DT_LONGVARCHAR	Length field ignored.	Length field ignored.
DT_NFIXCHAR(n)	Length of buffer, in bytes. The maximum value for n is 32767.	Padded with blanks to length of buffer.
DT_NSTRING	Length of buffer.	null character at end of string.
DT_NVARCHAR(n)	Maximum length of NVARCHAR structure (n+2). The maximum value for n is 32765.	len field of NVARCHAR structure set to actual length in bytes of string.
DT_SMALLINT	No action required.	No action required.
DT_STRING	Length of buffer.	null character at end of string.
DT_TIME	Length of buffer.	null character at end of string.
DT_TIMESTAMP	Length of buffer.	null character at end of string.
DT_TIMESTAMP_STRUCT	No action required.	No action required.
DT_UNSBIGINT	No action required.	No action required.
DT_UNSENT	No action required.	No action required.
DT_UNSSMALLINT	No action required.	No action required.



Embedded SQL data type	What the program must set length field to when receiving	How the database returns length information after fetching a value
DT_VARCHAR(n)	Maximum length of VARCHAR structure (n+2). The maximum value for n is 32765.	len field of VARCHAR structure set to actual length in bytes of string.

**See also**

- [“Retrieving LONG data” on page 488](#)

## Fetching data

Fetching data in embedded SQL is done using the SELECT statement. There are two cases:

- **The SELECT statement returns at most one row** Use an INTO clause to assign the returned values directly to host variables.
- **The SELECT statement may return multiple rows** Use cursors to manage the rows of the result set.

**See also**

- [“FETCH statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#)

## SELECT statements that return at most one row

A single row query retrieves at most one row from the database. A single-row query SELECT statement has an INTO clause following the SELECT list and before the FROM clause. The INTO clause contains a list of host variables to receive the value for each SELECT list item. There must be the same number of host variables as there are SELECT list items. The host variables may be accompanied by indicator variables to indicate NULL results.

When the SELECT statement is executed, the database server retrieves the results and places them in the host variables. If the query results contain more than one row, the database server returns an error.

If the query results in no rows being selected, an error is returned indicating that no rows can be found (SQLCODE 100). Errors and warnings are returned in the SQLCA structure.

**Example**

The following code fragment returns 1 if a row from the Employees table is fetched successfully, 0 if the row doesn't exist, and -1 if an error occurs.

```
EXEC SQL BEGIN DECLARE SECTION;
long      id;
char      name[41];
```

```
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
...
int find_employee( long employee_id )
{
    id = employee_id;
    EXEC SQL SELECT GivenName ||
        ' ' || Surname, Sex, BirthDate
        INTO :name, :sex,
            :birthdate:ind_birthdate
        FROM Employees
        WHERE EmployeeID = :id;
    if( SQLCODE == SQLE_NOTFOUND )
    {
        return( 0 ); /* employee not found */
    }
    else if( SQLCODE < 0 )
    {
        return( -1 ); /* error */
    }
    else
    {
        return( 1 ); /* found */
    }
}
```

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## Cursors in embedded SQL

A cursor is used to retrieve rows from a query that has multiple rows in its result set. A cursor is a handle or an identifier for the SQL query and a position within the result set.

### Manage a cursor in embedded SQL

1. Declare a cursor for a particular SELECT statement, using the DECLARE CURSOR statement.
2. Open the cursor using the OPEN statement.
3. Retrieve results one row at a time from the cursor using the FETCH statement.
4. Fetch rows until the Row Not Found warning is returned.

Errors and warnings are returned in the SQLCA structure.

5. Close the cursor, using the CLOSE statement.

By default, cursors are automatically closed at the end of a transaction (on COMMIT or ROLLBACK). Cursors that are opened with a WITH HOLD clause are kept open for subsequent transactions until they are explicitly closed.

The following is a simple example of cursor usage:

```
void print_employees( void )
{
EXEC SQL BEGIN DECLARE SECTION;
char      name[50];
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT GivenName || ' ' || Surname,
           Sex, BirthDate
    FROM Employees;
EXEC SQL OPEN C1;
for( ;; )
{
    EXEC SQL FETCH C1 INTO :name, :sex,
        :birthdate:ind_birthdate;
    if( SQLCODE == SQLE_NOTFOUND )
    {
        break;
    }
    else if( SQLCODE < 0 )
    {
        break;
    }

    if( ind_birthdate < 0 )
    {
        strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate:
        %s.n", name, sex, birthdate );
}
EXEC SQL CLOSE C1;
}
```

### Cursor positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row

Absolute row from start		Absolute row from end
0	Before first row	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		$-3$
$n - 1$		$-2$
$n$		$-1$
$n + 1$	After last row	0

When a cursor is opened, it is positioned before the first row. The cursor position can be moved using the FETCH statement. It can be positioned to an absolute position either from the start or from the end of the query results. It can also be moved relative to the current cursor position.

There are special **positioned** versions of the UPDATE and DELETE statements that can be used to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, an error is returned indicating that there is no corresponding row in the cursor.

The PUT statement can be used to insert a row into a cursor.

### Cursor positioning problems

Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.

With SQL Anywhere, this occurs if a temporary table had to be created to open the cursor.

The UPDATE statement can cause a row to move in the cursor. This happens if the cursor has an ORDER BY clause that uses an existing index (a temporary table is not created).

**See also**

- “CLOSE statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “DECLARE CURSOR statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “DELETE statement (positioned) [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “FETCH statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “OPEN statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “PUT statement [ESQL]” [*SQL Anywhere Server - SQL Reference*]
- “UPDATE (positioned) statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “Working with cursors” on page 7
- “Static cursor sample” on page 447
- “Dynamic cursor sample” on page 447
- “The SQL Communication Area (SQLCA)” on page 461
- “Use work tables in query processing (use All-rows optimization goal)” [*SQL Anywhere Server - SQL Usage*]

## Fetching more than one row at a time

The FETCH statement can be modified to fetch more than one row at a time, which may improve performance. This is called a **wide fetch** or an **array fetch**.

SQL Anywhere also supports wide puts and inserts.

To use wide fetches in embedded SQL, include the FETCH statement in your code as follows:

```
EXEC SQL FETCH ... ARRAY nnn
```

where ARRAY *nnn* is the last item of the FETCH statement. The fetch count *nnn* can be a host variable. The number of variables in the SQLDA must be the product of *nnn* and the number of columns per row. The first row is placed in SQLDA variables 0 to (columns per row) - 1, and so on.

Each column must be of the same type in each row of the SQLDA, or a SQLDA\_INCONSISTENT error is returned.

The server returns in SQLCOUNT the number of records that were fetched, which is always greater than zero unless there is an error or warning. On a wide fetch, a SQLCOUNT of 1 with no error condition indicates that one valid row has been fetched.

**Example**

The following example code illustrates the use of wide fetches. You can also find this code in %SQLANYSAMP12%\SQLAnywhere\esqlwidefetch\widefetch.sqc.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLException();
                             goto err; };
```

```

static void PrintSQLException()
{
    char buffer[200];

    printf( "SQL error %d -- %s\n",
            SQLCODE,
            sqlerror_message( &sqlca,
                             buffer,
                             sizeof( buffer ) ) );
}
static SQLDA * PrepareSQLDA(
    a_sql_statement_number stat0,
    unsigned width,
    unsigned *cols_per_row )

/* Allocate a SQLDA to be used for fetching from
   the statement identified by "stat0". "width"
   rows are retrieved on each FETCH request.
   The number of columns per row is assigned to
   "cols_per_row". */
{
    int          num_cols;
    unsigned     row, col, offset;
    SQLDA *      sqlda;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    EXEC SQL END DECLARE SECTION;
    stat = stat0;
    sqlda = alloc_sqlda( 100 );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
    *cols_per_row = num_cols = sqlda->sqld;
    if( num_cols * width > sqlda->sqln )
    {
        free_sqlda( sqlda );
        sqlda = alloc_sqlda( num_cols * width );
        if( sqlda == NULL ) return( NULL );
        EXEC SQL DESCRIBE :stat INTO sqlda;
    }
    // copy first row in SQLDA setup by describe
    // to following (wide) rows
    sqlda->sqld = num_cols * width;
    offset = num_cols;
    for( row = 1; row < width; row++ )
    {
        for( col = 0;
            col < num_cols;
            col++, offset++ )
        {
            sqlda->sqlvar[offset].sqltype =
                sqlda->sqlvar[col].sqltype;
            sqlda->sqlvar[offset].sqlllen =
                sqlda->sqlvar[col].sqlllen;
            // optional: copy described column name
            memcpy( &sqlda->sqlvar[offset].sqlname,
                   &sqlda->sqlvar[col].sqlname,
                   sizeof( sqlda->sqlvar[0].sqlname ) );
        }
    }
    fill_s_sqlda( sqlda, 40 );
    return( sqlda );
err:
    return( NULL );
}

```

```

static void PrintFetchedRows(
    SQLDA * sqlda,
    unsigned cols_per_row )
{
    /* Print rows already wide fetched in the SQLDA */
    long      rows_fetched;
    int       row, col, offset;

    if( SQLCOUNT == 0 )
    {
        rows_fetched = 1;
    }
    else
    {
        rows_fetched = SQLCOUNT;
    }
    printf( "Fetched %d Rows:\n", rows_fetched );
    for( row = 0; row < rows_fetched; row++ )
    {
        for( col = 0; col < cols_per_row; col++ )
        {
            offset = row * cols_per_row + col;
            printf( "  \"%s\"",
                (char *)sqlda->sqlvar[offset].sqldata );
        }
        printf( "\n" );
    }
}

static int DoQuery(
    char * query_str0,
    unsigned fetch_width0 )
{
    /* Wide Fetch "query_str0" select statement
     * using a width of "fetch_width0" rows */
    SQLDA *      sqlda;
    unsigned     cols_per_row;
    EXEC SQL BEGIN DECLARE SECTION;
    a_sql_statement_number stat;
    char *       query_str;
    unsigned     fetch_width;
    EXEC SQL END DECLARE SECTION;

    query_str = query_str0;
    fetch_width = fetch_width0;

    EXEC SQL PREPARE :stat FROM :query_str;
    EXEC SQL DECLARE QCURSOR CURSOR FOR :stat
        FOR READ ONLY;
    EXEC SQL OPEN QCURSOR;
    sqlda = PrepareSQLDA( stat,
        fetch_width,
        &cols_per_row );
    if( sqlda == NULL )
    {
        printf( "Error allocating SQLDA\n" );
        return( SQLE_NO_MEMORY );
    }
    for( ;; )
    {
        EXEC SQL FETCH QCURSOR INTO DESCRIPTOR sqlda
            ARRAY :fetch_width;
        if( SQLCODE != SQLE_NOERROR ) break;
        PrintFetchedRows( sqlda, cols_per_row );
    }
}

```

```

EXEC SQL CLOSE QCURSOR;
EXEC SQL DROP STATEMENT :stat;
free_filled_sqlda( sqlda );
err:
return( SQLCODE );
}
void main( int argc, char *argv[] )
{
/* Optional first argument is a select statement,
 * optional second argument is the fetch width */
char *query_str =
"SELECT GivenName, Surname FROM Employees";
unsigned fetch_width = 10;

if( argc > 1 )
{
query_str = argv[1];
if( argc > 2 )
{
fetch_width = atoi( argv[2] );
if( fetch_width < 2 )
{
fetch_width = 2;
}
}
}
db_init( &sqlca );
EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";

DoQuery( query_str, fetch_width );

EXEC SQL DISCONNECT;
err:
db_fini( &sqlca );
}

```

### Notes on using wide fetches

- In the function PrepareSQLDA, the SQLDA memory is allocated using the alloc\_sqlda function. This allows space for indicator variables, rather than using the alloc\_sqlda\_noind function.
- If the number of rows fetched is fewer than the number requested, but is not zero (at the end of the cursor for example), the SQLDA items corresponding to the rows that were not fetched are returned as NULL by setting the indicator value. If no indicator variables are present, an error is generated (SQLE\_NO\_INDICATOR: no indicator variable for NULL result).
- If a row being fetched has been updated, generating a SQLE\_ROW\_UPDATED\_WARNING warning, the fetch stops on the row that caused the warning. The values for all rows processed to that point (including the row that caused the warning) are returned. SQLCOUNT contains the number of rows that were fetched, including the row that caused the warning. All remaining SQLDA items are marked as NULL.
- If a row being fetched has been deleted or is locked, generating a SQLE\_NO\_CURRENT\_ROW or SQLE\_LOCKED error, SQLCOUNT contains the number of rows that were read before the error. This does not include the row that caused the error. The SQLDA does not contain values for any of the rows since SQLDA values are not returned on errors. The SQLCOUNT value can be used to reposition the cursor, if necessary, to read the rows.



**See also**

- “EXECUTE statement [ESQL]” [*SQL Anywhere Server - SQL Reference*]
- “FETCH statement [ESQL] [SP]” [*SQL Anywhere Server - SQL Reference*]
- “PUT statement [ESQL]” [*SQL Anywhere Server - SQL Reference*]
- “alloc\_sqlda function” on page 495
- “alloc\_sqlda\_noind function” on page 496

## Sending and retrieving long values

The method for sending and retrieving LONG VARCHAR, LONG NVARCHAR, and LONG BINARY values in embedded SQL applications is different from that for other data types. The standard SQLDA fields are limited to 32767 bytes of data as the fields holding the length information (sqlen, \*sqlind) are 16-bit values. Changing these values to 32-bit values would break existing applications.

The method of describing LONG VARCHAR, LONG NVARCHAR, and LONG BINARY values is the same as for other data types.

**Static SQL structures**

Separate fields are used to hold the allocated, stored, and untruncated lengths of LONG BINARY, LONG VARCHAR, and LONG NVARCHAR data types. The static SQL data types are defined in *sqlca.h* as follows:

```
#define DECL_LONGVARCHAR( size )      \
    struct { a_sql_uint32    array_len; \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size+1]; \
    }
#define DECL_LONGNVARCHAR( size )    \
    struct { a_sql_uint32    array_len; \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size+1]; \
    }
#define DECL_LONGBINARY( size )     \
    struct { a_sql_uint32    array_len; \
             a_sql_uint32    stored_len; \
             a_sql_uint32    untrunc_len; \
             char             array[size]; \
    }
```

**Dynamic SQL structures**

For dynamic SQL, set the sqltype field to DT\_LONGVARCHAR, DT\_LONGNVARCHAR, or DT\_LONGBINARY as appropriate. The associated LONGVARCHAR, LONGNVARCHAR, and LONGBINARY structure is as follows:

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

### Structure member definitions

For both static and dynamic SQL structures, the structure members are defined as follows:

- **array\_len** (Sending and retrieving.) The number of bytes allocated for the array part of the structure.
- **stored\_len** (Sending and retrieving.) The number of bytes stored in the array. Always less than or equal to `array_len` and `untrunc_len`.
- **untrunc\_len** (Retrieving only.) The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to `stored_len`. If truncation occurs, this value is larger than `array_len`.

## Retrieving LONG data

This section describes how to retrieve LONG values from the database.

The procedures are different depending on whether you are using static or dynamic SQL.

### Receive a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value (static SQL)

1. Declare a host variable of type `DECL_LONGVARCHAR`, `DECL_LONGNVARCHAR`, or `DECL_LONGBINARY`, as appropriate. The `array_len` member is filled in automatically.
2. Retrieve the data using `FETCH`, `GET DATA`, or `EXECUTE INTO`. SQL Anywhere sets the following information:
  - **indicator variable** Negative if the value is NULL, 0 if there is no truncation, otherwise the positive untruncated length in bytes up to a maximum of 32767.
  - **stored\_len** The number of bytes stored in the array. Always less than or equal to `array_len` and `untrunc_len`.
  - **untrunc\_len** The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to `stored_len`. If truncation occurs, this value is larger than `array_len`.

### Receive a value into a LONGVARCHAR, LONGNVARCHAR, or LONGBINARY structure (dynamic SQL)

1. Set the `sqltype` field to `DT_LONGVARCHAR`, `DT_LONGNVARCHAR`, or `DT_LONGBINARY` as appropriate.
2. Set the `sqldata` field to point to the `LONGVARCHAR`, `LONGNVARCHAR`, or `LONGBINARY` host variable structure.

You can use the `LONGVARCHARSIZE(n)`, `LONGNVARCHARSIZE(n)`, or `LONGBINARYSIZE(n)` macro to determine the total number of bytes to allocate to hold *n* bytes of data in the array field.

3. Set the `array_len` field of the host variable structure to the number of bytes allocated for the array field.
4. Retrieve the data using `FETCH`, `GET DATA`, or `EXECUTE INTO`. SQL Anywhere sets the following information:
  - **\* `sqlind`** This `sqlda` field is negative if the value is `NULL`, 0 if there is no truncation, and is the positive untruncated length in bytes up to a maximum of 32767.
  - **`stored_len`** The number of bytes stored in the array. Always less than or equal to `array_len` and `untrunc_len`.
  - **`untrunc_len`** The number of bytes that would be stored in the array if the value was not truncated. Always greater than or equal to `stored_len`. If truncation occurs, this value is larger than `array_len`.

The following code fragment illustrates the mechanics of retrieving `LONG VARCHAR` data using dynamic embedded SQL. It is not intended to be a practical application:

```
#define DATA_LEN 128000
void get_test_var()
{
    LONGVARCHAR *longptr;
    SQLDA      *sqlda;
    SQLVAR      *sqlvar;

    sqlda = alloc_sqlda( 1 );
    longptr = (LONGVARCHAR *)malloc(
        LONGVARCHARSIZE( DATA_LEN ) );
    if( sqlda == NULL || longptr == NULL )
    {
        fatal_error( "Allocation failed" );
    }

    // init longptr for receiving data
    longptr->array_len = DATA_LEN;

    // init sqlda for receiving data
    // (sqlen is unused with DT_LONG types)
    sqlda->sqld = 1; // using 1 sqlvar
    sqlvar = &sqlda->sqlvar[0];
    sqlvar->sqltype = DT_LONGVARCHAR;
    sqlvar->sqldata = longptr;
    printf( "fetching test_var\n" );
    EXEC SQL PREPARE select_stmt FROM 'SELECT test_var';
    EXEC SQL EXECUTE select_stmt INTO DESCRIPTOR sqlda;
    EXEC SQL DROP STATEMENT select_stmt;
    printf( "stored_len: %d, untrunc_len: %d, "
        "1st char: %c, last char: %c\n",
        longptr->stored_len,
        longptr->untrunc_len,
        longptr->array[0],
        longptr->array[DATA_LEN - 1] );
    free_sqlda( sqlda );
    free( longptr );
}
```

**See also**

- [“Sending and retrieving long values” on page 487](#)
- [“Indicator variables” on page 458](#)

## Sending LONG data

This section describes how to send LONG values to the database from embedded SQL applications.

The procedures are different depending on whether you are using static or dynamic SQL.

### Send a LONG value (static SQL)

1. Declare a host variable of type DECL\_LONGVARCHAR, DECL\_LONGNVARCHAR, or DECL\_LONGBINARY, as appropriate.
2. If you are sending NULL, set the indicator variable to a negative value.
3. Set the stored\_len field of the host variable structure to the number of bytes of data in the array field.
4. Send the data by opening the cursor or executing the statement.

The following code fragment illustrates the mechanics of sending a LONG VARCHAR using static embedded SQL. It is not intended to be a practical application.

```
#define DATA_LEN 12800
EXEC SQL BEGIN DECLARE SECTION;
// SQLPP initializes longdata.array_len
DECL_LONGVARCHAR(128000) longdata;
EXEC SQL END DECLARE SECTION;

void set_test_var()
{
    // init longdata for sending data
    memset( longdata.array, 'a', DATA_LEN );
    longdata.stored_len = DATA_LEN;

    printf( "Setting test_var to %d a's\n", DATA_LEN );
    EXEC SQL SET test_var = :longdata;
}
```

### Send a LONG value (dynamic SQL)

1. Set the sqltype field to DT\_LONGVARCHAR, DT\_LONGNVARCHAR, or DT\_LONGBINARY, as appropriate.
2. If you are sending NULL, set \* **sqlind** to a negative value.
3. If you are not sending NULL, set the sqldata field to point to the LONGVARCHAR, LONGNVARCHAR, or LONGBINARY host variable structure.

You can use the `LONGVARCHARSIZE(n)`, `LONGNVARCHARSIZE(n)`, or `LONGBINARYSIZE(n)` macros to determine the total number of bytes to allocate to hold *n* bytes of data in the array field.

4. Set the `array_len` field of the host variable structure to the number of bytes allocated for the array field.
5. Set the `stored_len` field of the host variable structure to the number of bytes of data in the array field. This must not be more than `array_len`.
6. Send the data by opening the cursor or executing the statement.

**See also**

- [“Sending and retrieving long values” on page 487](#)
- [“Indicator variables” on page 458](#)

## Simple stored procedures in embedded SQL

You can create and call stored procedures in embedded SQL.

You can embed a `CREATE PROCEDURE` just like any other data definition statement, such as `CREATE TABLE`. You can also embed a `CALL` statement to execute a stored procedure. The following code fragment illustrates both creating and executing a stored procedure in embedded SQL:

```
EXEC SQL CREATE PROCEDURE pettycash(  
    IN Amount DECIMAL(10,2) )  
BEGIN  
    UPDATE account  
    SET balance = balance - Amount  
    WHERE name = 'bank';  
  
    UPDATE account  
    SET balance = balance + Amount  
    WHERE name = 'pettycash expense';  
END;  
EXEC SQL CALL pettycash( 10.72 );
```

If you want to pass host variable values to a stored procedure or to retrieve the output variables, you prepare and execute a `CALL` statement. The following code fragment illustrates the use of host variables. Both the `USING` and `INTO` clauses are used on the `EXECUTE` statement.

```
EXEC SQL BEGIN DECLARE SECTION;  
double hv_expense;  
double hv_balance;  
EXEC SQL END DECLARE SECTION;  
  
// Code here  
EXEC SQL CREATE PROCEDURE pettycash(  
    IN expense DECIMAL(10,2),  
    OUT endbalance DECIMAL(10,2) )  
BEGIN  
    UPDATE account  
    SET balance = balance - expense
```

```
WHERE name = 'bank';
UPDATE account
SET balance = balance + expense
WHERE name = 'pettycash expense';

SET endbalance = ( SELECT balance FROM account
                   WHERE name = 'bank' );
END;

EXEC SQL PREPARE S1 FROM 'CALL pettycash( ?, ? )';
EXEC SQL EXECUTE S1 USING :hv_expense INTO :hv_balance;
```

### See also

- “CREATE PROCEDURE statement” [[SQL Anywhere Server - SQL Reference](#)]
- “EXECUTE statement [ESQL]” [[SQL Anywhere Server - SQL Reference](#)]
- “PREPARE statement [ESQL]” [[SQL Anywhere Server - SQL Reference](#)]

## Stored procedures with result sets

Database procedures can also contain SELECT statements. The procedure is declared using a RESULT clause to specify the number, name, and types of the columns in the result set. Result set columns are different from output parameters. For procedures with result sets, the CALL statement can be used in place of a SELECT statement in the cursor declaration:

```
EXEC SQL BEGIN DECLARE SECTION;
char hv_name[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE PROCEDURE female_employees()
RESULT( name char(50) )
BEGIN
SELECT GivenName || Surname FROM Employees
WHERE Sex = 'f';
END;

EXEC SQL PREPARE S1 FROM 'CALL female_employees()';

EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
for(;;)
{
EXEC SQL FETCH C1 INTO :hv_name;
if( SQLCODE != SQLE_NOERROR ) break;
printf( "%s\n", hv_name );
}
EXEC SQL CLOSE C1;
```

In this example, the procedure has been invoked with an OPEN statement rather than an EXECUTE statement. The OPEN statement causes the procedure to execute until it reaches a SELECT statement. At this point, C1 is a cursor for the SELECT statement within the database procedure. You can use all forms of the FETCH statement (backward and forward scrolling) until you are finished with it. The CLOSE statement stops execution of the procedure.

If there had been another statement following the SELECT in the procedure, it would not have been executed. To execute statements following a SELECT, use the RESUME cursor-name statement. The

RESUME statement either returns the warning `SQLCODE = SQLSTATE = SQLE_PROCEDURE_COMPLETE` or it returns `SQLCODE = SQLSTATE = SQLE_NOERROR` indicating that there is another cursor. The example illustrates a two-select procedure:

```
EXEC SQL CREATE PROCEDURE people()
  RESULT( name char(50) )
BEGIN
  SELECT GivenName || Surname
  FROM Employees;

  SELECT GivenName || Surname
  FROM Customers;
END;

EXEC SQL PREPARE S1 FROM 'CALL people()';
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
while( SQLCODE == SQLE_NOERROR )
{
  for(;;)
  {
    EXEC SQL FETCH C1 INTO :hv_name;
    if( SQLCODE != SQLE_NOERROR ) break;
    printf( "%s\n", hv_name );
  }
  EXEC SQL RESUME C1;
}
EXEC SQL CLOSE C1;
```

### Dynamic cursors for CALL statements

These examples have used static cursors. Full dynamic cursors can also be used for the CALL statement.

The DESCRIBE statement works fully for procedure calls. A DESCRIBE OUTPUT produces a SQLDA that has a description for each of the result set columns.

If the procedure does not have a result set, the SQLDA has a description for each INOUT or OUT parameter for the procedure. A DESCRIBE INPUT statement produces a SQLDA having a description for each IN or INOUT parameter for the procedure.

### DESCRIBE ALL

DESCRIBE ALL describes IN, INOUT, OUT, and RESULT set parameters. DESCRIBE ALL uses the indicator variables in the SQLDA to provide additional information.

The `DT_PROCEDURE_IN` and `DT_PROCEDURE_OUT` bits are set in the indicator variable when a CALL statement is described. `DT_PROCEDURE_IN` indicates an IN or INOUT parameter and `DT_PROCEDURE_OUT` indicates an INOUT or OUT parameter. Procedure RESULT columns have both bits clear.

After a DESCRIBE OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE).

### Multiple result sets

If you have a procedure that returns multiple result sets, you must re-describe after each RESUME statement if the result sets change shapes.

You need to describe the cursor, not the statement, to re-describe the current position of the cursor.

### See also

- [“CREATE PROCEDURE statement” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Dynamic SELECT statement” on page 468](#)
- [“DESCRIBE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Request management with embedded SQL

Since a typical embedded SQL application must wait for the completion of each database request before carrying out the next step, an application that uses multiple execution threads can carry on with other tasks.

If you must use a single execution thread, then some degree of multitasking can be accomplished by registering a callback function using the `db_register_a_callback` function with the `DB_CALLBACK_WAIT` option. Your callback function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

In your callback function, you cannot start another database request but you can cancel the current request using the `db_cancel_request` function. You can use the `db_is_working` function in your message handlers to determine if you have a database request in progress.

### See also

- [“db\\_register\\_a\\_callback function” on page 510](#)
- [“db\\_cancel\\_request function” on page 501](#)
- [“db\\_is\\_working function” on page 507](#)

## Database backup with embedded SQL

The recommended way to backup a database is to use the `BACKUP` statement.

The `db_backup` function provides another way to perform an online backup in embedded SQL applications. Note that the SQL Anywhere Backup utility also makes use of this function.

You can also interface directly to the SQL Anywhere Backup utility using the Database Tools `DBBackup` function.

You should only undertake to write a program using the `db_backup` function if your backup requirements are not satisfied by the any of the other backup methods.



**See also**

- “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- “Backup utility (dbbackup)” [*SQL Anywhere Server - Database Administration*]
- “DBBackup method [database tools]” on page 853
- “db\_backup function” on page 496

## Library function reference

The SQL preprocessor generates calls to functions in the interface library or DLL. In addition to the calls generated by the SQL preprocessor, a set of library functions is provided to make database operations easier to perform. Prototypes for these functions are included by the EXEC SQL INCLUDE SQLCA statement.

This section contains a reference description of these various functions.

**DLL entry points**

The DLL entry points are the same except that the prototypes have a modifier appropriate for DLLs.

You can declare the entry points in a portable manner using `_esqlentry_`, which is defined in *sqlca.h*. It resolves to the value `__stdcall`.

## alloc\_sqllda function

**Syntax**

```
struct sqllda * alloc_sqllda( unsigned numvar );
```

**Parameters**

- **numvar** The number of variable descriptors to allocate.

**Returns**

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

**Remarks**

Allocates a SQLDA with descriptors for *numvar* variables. The *sqln* field of the SQLDA is initialized to *numvar*. Space is allocated for the indicator variables, the indicator pointers are set to point to this space, and the indicator value is initialized to zero. A null pointer is returned if memory cannot be allocated. It is recommended that you use this function instead of the `alloc_sqllda_noind` function.

### See also

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)
- [“fill\\_s\\_sqlda function” on page 519](#)
- [“fill\\_sqlda function” on page 520](#)
- [“fill\\_sqlda\\_ex function” on page 520](#)
- [“free\\_sqlda function” on page 522](#)

## alloc\_sqlda\_noind function

### Syntax

```
struct sqlda * alloc_sqlda_noind( unsigned numvar );
```

### Parameters

- **numvar** The number of variable descriptors to allocate.

### Returns

Pointer to a SQLDA if successful and returns the null pointer if there is not enough memory available.

### Remarks

Allocates a SQLDA with descriptors for *numvar* variables. The *sqln* field of the SQLDA is initialized to *numvar*. Space is not allocated for indicator variables; the indicator pointers are set to the null pointer. A null pointer is returned if memory cannot be allocated.

### See also

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)
- [“fill\\_s\\_sqlda function” on page 519](#)
- [“fill\\_sqlda function” on page 520](#)
- [“fill\\_sqlda\\_ex function” on page 520](#)
- [“free\\_sqlda\\_noind function” on page 522](#)

## db\_backup function

### Syntax

```
void db_backup(  
SQLCA * sqlca,  
int op,  
int file_num,  
unsigned long page_num,  
struct sqlda * sqlda);
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.

- **op** The action or operation to be performed.
- **file\_num** The file number of the database.
- **page\_num** The page number of the database. A value in the range 0 to the maximum number of pages less 1.
- **sqlda** A pointer to a SQLDA structure.

### Authorization

Must be connected as a user with DBA authority, REMOTE DBA authority (SQL Remote), or BACKUP authority.

### Remarks

Although this function provides one way to add backup features to an application, the recommended way to do this task is to use the BACKUP statement.

The action performed depends on the value of the *op* parameter:

- **DB\_BACKUP\_START** Must be called before a backup can start. Only one backup can be running per database at one time against any given database server. Database checkpoints are disabled until the backup is complete (db\_backup is called with an *op* value of DB\_BACKUP\_END). If the backup cannot start, the SQLCODE is SQLE\_BACKUP\_NOT\_STARTED. Otherwise, the SQLCOUNT field of the *sqlca* is set to the database page size. Backups are processed one page at a time.

The *file\_num*, *page\_num*, and *sqlda* parameters are ignored.

- **DB\_BACKUP\_OPEN\_FILE** Open the database file specified by *file\_num*, which allows pages of the specified file to be backed up using DB\_BACKUP\_READ\_PAGE. Valid file numbers are 0 through DB\_BACKUP\_MAX\_FILE for the root database files, and 0 through DB\_BACKUP\_TRANS\_LOG\_FILE for the transaction log file. If the specified file does not exist, the SQLCODE is SQLE\_NOTFOUND. Otherwise, SQLCOUNT contains the number of pages in the file, SQLIOESTIMATE contains a 32-bit value (POSIX time\_t) that identifies the time that the database file was created, and the operating system file name is in the *sqlerrmc* field of the SQLCA.

The *page\_num* and *sqlda* parameters are ignored.

- **DB\_BACKUP\_READ\_PAGE** Read one page of the database file specified by *file\_num*. The *page\_num* should be a value from 0 to one less than the number of pages returned in SQLCOUNT by a successful call to db\_backup with the DB\_BACKUP\_OPEN\_FILE operation. Otherwise, SQLCODE is set to SQLE\_NOTFOUND. The *sqlda* descriptor should be set up with one variable of type DT\_BINARY or DT\_LONG\_BINARY pointing to a buffer. The buffer should be large enough to hold binary data of the size returned in the SQLCOUNT field on the call to db\_backup with the DB\_BACKUP\_START operation.

DT\_BINARY data contains a two-byte length followed by the actual binary data, so the buffer must be two bytes longer than the page size.

**Note**

This call makes a copy of the specified database page into the buffer, but it is up to the application to save the buffer on some backup media.

- **DB\_BACKUP\_READ\_RENAME\_LOG** This action is the same as `DB_BACKUP_READ_PAGE`, except that after the last page of the transaction log has been returned, the database server renames the transaction log and starts a new one.

If the database server is unable to rename the log at the current time (for example in version 7.0.x or earlier databases there may be incomplete transactions), the `SQLC_BACKUP_CANNOT_RENAME_LOG_YET` error is set. In this case, do not use the page returned, but instead reissue the request until you receive `SQLC_NOERROR` and then write the page. Continue reading the pages until you receive the `SQLC_NOTFOUND` condition.

The `SQLC_BACKUP_CANNOT_RENAME_LOG_YET` error may be returned multiple times and on multiple pages. In your retry loop, you should add a delay so as not to slow the server down with too many requests.

When you receive the `SQLC_NOTFOUND` condition, the transaction log has been backed up successfully and the file has been renamed. The name for the old transaction file is returned in the `sqlerrmc` field of the `SQLCA`.

You should check the `sqlca->sqlvar[0].sqlind` value after a `db_backup` call. If this value is greater than zero, the last log page has been written and the log file has been renamed. The new name is still in `sqlca.sqlerrmc`, but the `SQLCODE` value is `SQLC_NOERROR`.

You should not call `db_backup` again after this, except to close files and finish the backup. If you do, you get a second copy of your backed up log file and you receive `SQLC_NOTFOUND`.

- **DB\_BACKUP\_CLOSE\_FILE** Must be called when processing of one file is complete to close the database file specified by `file_num`.

The `page_num` and `sqlca` parameters are ignored.

- **DB\_BACKUP\_END** Must be called at the end of the backup. No other backup can start until this backup has ended. Checkpoints are enabled again.

The `file_num`, `page_num` and `sqlca` parameters are ignored.

- **DB\_BACKUP\_PARALLEL\_START** Starts a parallel backup. Like `DB_BACKUP_START`, only one backup can be running against a database at one time on any given database server. Database checkpoints are disabled until the backup is complete (until `db_backup` is called with an `op` value of `DB_BACKUP_END`). If the backup cannot start, you receive `SQLC_BACKUP_NOT_STARTED`. Otherwise, the `SQLCOUNT` field of the `sqlca` is set to the database page size.

The `file_num` parameter instructs the database server to rename the transaction log and start a new one after the last page of the transaction log has been returned. If the value is non-zero then the transaction log is renamed or restarted. Otherwise, it is not renamed and restarted. This parameter eliminates the need for the `DB_BACKUP_READ_RENAME_LOG` operation, which is not allowed during a parallel backup operation.

The *page\_num* parameter informs the database server of the maximum size of the client's buffer, in database pages. On the server side, the parallel backup readers try to read sequential blocks of pages—this value lets the server know how large to allocate these blocks: passing a value of *nnn* lets the server know that the client is willing to accept at most *nnnn* database pages at a time from the server. The server may return blocks of pages of less than size *nnn* if it is unable to allocate enough memory for blocks of *nnn* pages. If the client does not know the size of database pages until after the call to `DB_BACKUP_PARALLEL_START`, this value can be provided to the server with the `DB_BACKUP_INFO` operation. This value must be provided before the first call to retrieve backup pages (`DB_BACKUP_PARALLEL_READ`).

**Note**

If you are using `db_backup` to start a parallel backup, `db_backup` does not create writer threads. The caller of `db_backup` must receive the data and act as the writer.

- **DB\_BACKUP\_INFO** This parameter provides additional information to the database server about the parallel backup. The *file\_num* parameter indicates the type of information being provided, and the *page\_num* parameter provides the value. You can specify the following additional information with `DB_BACKUP_INFO`:
  - **DB\_BACKUP\_INFO\_PAGES\_IN\_BLOCK** The *page\_num* argument contains the maximum number of pages that should be sent back in one block.
  - **DB\_BACKUP\_INFO\_CHKPT\_LOG** This is the client-side equivalent to the `WITH CHECKPOINT LOG` option of the `BACKUP` statement. A *page\_num* value of `DB_BACKUP_CHKPT_COPY` indicates `COPY`, while the value `DB_BACKUP_CHKPT_NOCOPY` indicates `NO COPY`. If this value is not provided it defaults to `COPY`.
- **DB\_BACKUP\_PARALLEL\_READ** This operation reads a block of pages from the database server. Before invoking this operation, use the `DB_BACKUP_OPEN_FILE` operation to open all the files that you want to back up. `DB_BACKUP_PARALLEL_READ` ignores the *file\_num* and *page\_num* arguments.

The `sqlda` descriptor should be set up with one variable of type `DT_LONGBINARY` pointing to a buffer. The buffer should be large enough to hold binary data of the size *nnn* pages (specified in the `DB_BACKUP_START_PARALLEL` operation, or in a `DB_BACKUP_INFO` operation).

The server returns a sequential block of database pages for a particular database file. The page number of the first page in the block is returned in the `SQLCOUNT` field. The file number that the pages belong to is returned in the `SQLIOESTIMATE` field, and this value matches one of the file numbers used in the `DB_BACKUP_OPEN_FILE` calls. The size of the data returned is available in the *stored\_len* field of the `DT_LONGBINARY` variable, and is always a multiple of the database page size. While the data returned by this call contains a block of sequential pages for a given file, it is not safe to assume that separate blocks of data are returned in sequential order, or that all of one database file's pages are returned before another database file's pages. The caller should be prepared to receive portions of another individual file out of sequential order, or of any opened database file on any given call.

An application should make repeated calls to this operation until the size of the read data is 0, or the value of `sqlda->sqlvar[0].sqlind` is greater than 0. If the backup is started with transaction log renaming/restarting, `SQLERROR` could be set to `SQLE_BACKUP_CANNOT_RENAME_LOG_YET`. In this case, do not use the pages returned, but instead reissue the request until you receive `SQLE_NOERROR`, and then write the data. The `SQLE_BACKUP_CANNOT_RENAME_LOG_YET` error may be returned multiple times and on multiple pages. In your retry loop, you should add a delay so the database server is not slowed down by too many requests. Continue reading the pages until either of the first two conditions are met.

The `dbbackup` utility uses the following algorithm. Note that this is *not* C code, and does not include error checking.

```

sqlda->sqld = 1;
sqlda->sqlvar[0].sqltype = DT_LONGBINARY

/* Allocate LONGBINARY value for page buffer. It MUST have */
/* enough room to hold the requested number (128) of database pages */
sqlda->sqlvar[0].sqldata = allocated buffer

/* Open the server files needing backup */
for file_num = 0 to DB_BACKUP_MAX_FILE
  db_backup( ... DB_BACKUP_OPEN_FILE, file_num ... )
  if SQLCODE == SQLE_NO_ERROR
    /* The file exists */
    num_pages = SQLCOUNT
    file_time = SQLE_IO_ESTIMATE
    open backup file with name from sqlca.sqlerrmc
  end for

/* read pages from the server, write them locally */
while TRUE
  /* file_no and page_no are ignored */
  db_backup( &sqlca, DB_BACKUP_PARALLEL_READ, 0, 0, &sqlda );

  if SQLCODE != SQLE_NO_ERROR
    break;

  if buffer->stored_len == 0 || sqlda->sqlvar[0].sqlind > 0
    break;

  /* SQLCOUNT contains the starting page number of the block */
  /* SQLIOESTIMATE contains the file number the pages belong to */
  write block of pages to appropriate backup file
end while

/* close the server backup files */
for file_num = 0 to DB_BACKUP_MAX_FILE
  /* close backup file */
  db_backup( ... DB_BACKUP_CLOSE_FILE, file_num ... )
end for

/* shut down the backup */
db_backup( ... DB_BACKUP_END ... )

/* cleanup */
free page buffer

```

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“The SQL descriptor area \(SQLDA\)” on page 469](#)
- [“BACKUP statement” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Embedded SQL data types” on page 449](#)

## db\_cancel\_request function

**Syntax**

```
int db_cancel_request( SQLCA * sqlca );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.

**Returns**

1 when the cancel request is sent; 0 if no request is sent.

**Remarks**

Cancels the currently active database server request. This function checks to make sure a database server request is active before sending the cancel request.

A non-zero return value does not mean that the request was canceled. There are a few critical timing cases where the cancel request and the response from the database or server cross. In these cases, the cancel simply has no effect, even though the function still returns TRUE.

The `db_cancel_request` function can be called asynchronously. This function and `db_is_working` are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

If you cancel a request that is carrying out a cursor operation, the position of the cursor is indeterminate. You must locate the cursor by its absolute position or close it, following the cancel.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## db\_change\_char\_charset function

**Syntax**

```
unsigned int db_change_char_charset(  
SQLCA * sqlca,  
char * charset );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **charset** A string representing the character set.

### Returns

1 if the change is successful; 0 otherwise.

### Remarks

Changes the application's CHAR character set for this connection. Data sent and fetched using DT\_FIXCHAR, DT\_VARCHAR, DT\_LONGVARCHAR, and DT\_STRING types are in the CHAR character set.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Recommended character sets and collations” \[SQL Anywhere Server - Database Administration\]](#)

## db\_change\_nchar\_charset function

### Syntax

```
unsigned int db_change_nchar_charset(  
SQLCA * sqlca,  
char * charset );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **charset** A string representing the character set.

### Returns

1 if the change is successful; 0 otherwise.

### Remarks

Changes the application's NCHAR character set for this connection. Data sent and fetched using DT\_NFIXCHAR, DT\_NVARCHAR, DT\_LONGNVARCHAR, and DT\_NSTRING host variable types are in the NCHAR character set.

If the db\_change\_nchar\_charset function is not called, all data is sent and fetched using the CHAR character set. Typically, an application that wants to send and fetch Unicode data should set the NCHAR character set to UTF-8.

If this function is called, the charset parameter is usually "UTF-8". The NCHAR character set cannot be set to UTF-16.

In embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT\_FIXCHAR, DT\_VARCHAR, and DT\_LONGVARCHAR, respectively, by default. If the db\_change\_nchar\_charset



function has been called, these types are described as DT\_NFIXCHAR, DT\_NVARCHAR, and DT\_LONGNVARCHAR, respectively.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Recommended character sets and collations” \[\*SQL Anywhere Server - Database Administration\*\]](#)

## db\_delete\_file function

**Syntax**

```
void db_delete_file(  
SQLCA * sqlca,  
char * filename );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **filename** A string containing the path of the file.

**Authorization**

Must be connected to a user ID with DBA authority or REMOTE DBA authority (SQL Remote).

**Remarks**

The db\_delete\_file function requests the database server to delete *filename*. This can be used after backing up and renaming the transaction log to delete the old transaction log. Refer to DB\_BACKUP\_READ\_RENAME\_LOG of the db\_backup function.

You must be connected to a user ID with DBA authority.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“db\\_backup function” on page 496](#)

## db\_find\_engine function

**Syntax**

```
unsigned short db_find_engine(  
SQLCA * sqlca,  
char * name );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **name** NULL or a string containing the server's name.

## Returns

Server status as an unsigned short value, or 0 if no server can be found over shared memory.

## Remarks

Returns an unsigned short value, which indicates status information about the local database server whose name is *name*. If no server can be found over shared memory with the specified name, the return value is 0. A non-zero value indicates that the local server is currently running.

If a null pointer is specified for *name*, information is returned about the default database server.

Each bit in the return value conveys some information. Constants that represent the bits for the various pieces of information are defined in the *sqldef.h* header file. Their meaning is described below.

- **DB\_ENGINE** This flag is always set.
- **DB\_CLIENT** This flag is always set.
- **DB\_CAN\_MULTI\_DB\_NAME** This flag is obsolete.
- **DB\_DATABASE\_SPECIFIED** This flag is always set.
- **DB\_ACTIVE\_CONNECTION** This flag is always set.
- **DB\_CONNECTION\_DIRTY** This flag is obsolete.
- **DB\_CAN\_MULTI\_CONNECT** This flag is obsolete.
- **DB\_NO\_DATABASES** This flag is set if the server has no databases started.

## See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## db\_fini function

### Syntax

```
int db_fini( SQLCA * sqlca );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.

### Returns

Non-zero value for success; 0 otherwise.

### Remarks

This function frees resources used by the database interface or DLL. You must not make any other library calls or execute any embedded SQL statements after `db_fini` is called. If an error occurs during

processing, the error code is set in `SQLCA` and the function returns 0. If there are no errors, a non-zero value is returned.

You need to call `db_fini` once for each `SQLCA` being used.

Note that `db_fini` should not be called directly or indirectly from the `DllMain` function in a Windows Dynamic Link Library. The `DllMain` entry point function is intended to perform only simple initialization and termination tasks. Calling `db_fini` can create deadlocks and circular dependencies.

For UltraLite applications, there is an equivalent `db_fini` method.

### See also

- “The SQL Communication Area (SQLCA)” on page 461
- “`db_fini` method” [*UltraLite - C and C++ Programming*]

## db\_get\_property function

### Syntax

```
unsigned int db_get_property(  
SQLCA * sqlca,  
a_db_property property,  
char * value_buffer,  
int value_buffer_size );
```

### Parameters

- **sqlca** A pointer to a `SQLCA` structure.
- **a\_db\_property** The property requested, either `DB_PROP_CLIENT_CHARSET`, `DB_PROP_SERVER_ADDRESS`, or `DB_PROP_DBLIB_VERSION`.
- **value\_buffer** This argument is filled with the property value as a null-terminated string.
- **value\_buffer\_size** The maximum length of the string `value_buffer`, including room for the terminating null character.

### Returns

1 if successful; 0 otherwise.

### Remarks

This function is used to obtain information about the database interface or the server to which you are connected.

The following properties are supported:

- **DB\_PROP\_CLIENT\_CHARSET** This property value gets the client character set (for example, "windows-1252").

- **DB\_PROP\_SERVER\_ADDRESS** This property value gets the current connection's server network address as a printable string. The shared memory protocol always returns the empty string for the address. The TCP/IP protocol returns non-empty string addresses.
- **DB\_PROP\_DBLIB\_VERSION** This property value gets the database interface library's version (for example, "12.0.1.1297").

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## db\_init function

**Syntax**

```
int db_init( SQLCA * sqlca );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.

**Returns**

Non-zero value if successful; 0 otherwise.

**Remarks**

This function initializes the database interface library. This function must be called before any other library call is made and before any embedded SQL statement is executed. The resources the interface library required for your program are allocated and initialized on this call.

Use db\_fini to free the resources at the end of your program. If there are any errors during processing, they are returned in the SQLCA and 0 is returned. If there are no errors, a non-zero value is returned and you can begin using embedded SQL statements and functions.

Usually, this function should be called only once (passing the address of the global sqlca variable defined in the *sqlca.h* header file). If you are writing a DLL or an application that has multiple threads using embedded SQL, call db\_init once for each SQLCA that is being used.

For UltraLite applications, there is an equivalent db\_init method.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“SQLCA management for multithreaded or reentrant code” on page 463](#)
- [“db\\_init method” \[UltraLite - C and C++ Programming\]](#)

## db\_is\_working function

### Syntax

```
unsigned short db_is_working( SQLCA * sqlca );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.

### Returns

1 if your application has a database request in progress that uses the given *sqlca* and 0 if there is no request in progress that uses the given *sqlca*.

### Remarks

This function can be called asynchronously. This function and `db_cancel_request` are the only functions in the database interface library that can be called asynchronously using a SQLCA that might be in use by another request.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## db\_locate\_servers function

### Syntax

```
unsigned int db_locate_servers(  
SQLCA * sqlca,  
SQL_CALLBACK_PARM callback_address,  
void * callback_user_data );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **callback\_address** The address of a callback function.
- **callback\_user\_data** The address of a user-defined area in which to store data.

### Returns

1 if successful; 0 otherwise.

### Remarks

Provides programmatic access to the information displayed by the `dblocate` utility, listing all the SQL Anywhere database servers on the local network that are listening on TCP/IP.

The callback function must have the following prototype:

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

The callback function is called for each server found. If the callback function returns 0, `db_locate_servers` stops iterating through servers.

The `sqlca` and `callback_user_data` passed to the callback function are those passed into `db_locate_servers`. The second parameter is a pointer to an `a_server_address` structure. `a_server_address` is defined in `sqlca.h`, with the following definition:

```
typedef struct a_server_address {
    a_sql_uint32 port_type;
    a_sql_uint32 port_num;
    char *name;
    char *address;
} a_server_address;
```

- **port\_type** Is always `PORT_TYPE_TCP` at this time (defined to be 6 in `sqlca.h`).
- **port\_num** Is the TCP port number on which this server is listening.
- **name** Points to a buffer containing the server name.
- **address** Points to a buffer containing the IP address of the server.

#### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Server Enumeration utility \(dblocate\)” \[SQL Anywhere Server - Database Administration\]](#)

## db\_locate\_servers\_ex function

#### Syntax

```
unsigned int db_locate_servers_ex(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data,
unsigned int bitmask);
```

#### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **callback\_address** The address of a callback function.
- **callback\_user\_data** The address of a user-defined area in which to store data.
- **bitmask** A mask composed of any of `DB_LOOKUP_FLAG_NUMERIC`, `DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT`, or `DB_LOOKUP_FLAG_DATABASES`.

## Returns

1 if successful; 0 otherwise.

## Remarks

Provides programmatic access to the information displayed by the `dblocate` utility, listing all the SQL Anywhere database servers on the local network that are listening on TCP/IP, and provides a mask parameter used to select addresses passed to the callback function.

The callback function must have the following prototype:

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

The callback function is called for each server found. If the callback function returns 0, `db_locate_servers_ex` stops iterating through servers.

The `sqlca` and `callback_user_data` passed to the callback function are those passed into `db_locate_servers`. The second parameter is a pointer to an `a_server_address` structure. `a_server_address` is defined in `sqlca.h`, with the following definition:

```
typedef struct a_server_address {
    a_sql_uint32    port_type;
    a_sql_uint32    port_num;
    char            *name;
    char            *address;
    char            *dbname;
} a_server_address;
```

- **port\_type** Is always `PORT_TYPE_TCP` at this time (defined to be 6 in `sqlca.h`).
- **port\_num** Is the TCP port number on which this server is listening.
- **name** Points to a buffer containing the server name.
- **address** Points to a buffer containing the IP address of the server.
- **dbname** Points to a buffer containing the database name.

Three bitmask flags are supported:

- `DB_LOOKUP_FLAG_NUMERIC`
- `DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT`
- `DB_LOOKUP_FLAG_DATABASES`

These flags are defined in `sqlca.h` and can be ORed together.

`DB_LOOKUP_FLAG_NUMERIC` ensures that addresses passed to the callback function are IP addresses, instead of host names.

`DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT` specifies that the address includes the TCP/IP port number in the `a_server_address` structure passed to the callback function.

DB\_LOOKUP\_FLAG\_DATABASES specifies that the callback function is called once for each database found, or once for each database server found if the database server doesn't support sending database information (version 9.0.2 and earlier database servers).

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Server Enumeration utility \(dblocate\)” \[SQL Anywhere Server - Database Administration\]](#)

## db\_register\_a\_callback function

**Syntax**

```
void db_register_a_callback(
    SQLCA * sqlca,
    a_db_callback_index index,
    ( SQL_CALLBACK_PARM ) callback );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **index** An index value identifying the type of callback as described below.
- **callback** The address of a user-defined callback function.

**Remarks**

This function registers callback functions.

If you do not register a DB\_CALLBACK\_WAIT callback, the default action is to do nothing. Your application blocks, waiting for the database response. You must register a callback for the MESSAGE TO CLIENT statement.

To remove a callback, pass a null pointer as the *callback* function.

The following values are allowed for the *index* parameter:

- **DB\_CALLBACK\_DEBUG\_MESSAGE** The supplied function is called once for each debug message and is passed a null-terminated string containing the text of the debug message. A debug message is a message that is logged to the LogFile file. In order for a debug message to be passed to this callback, the LogFile connection parameter must be used. The string normally has a newline character (\n) immediately before the terminating null character. The prototype of the callback function is as follows:

```
void SQL_CALLBACK debug_message_callback(
    SQLCA * sqlca,
    char * message_string );
```

- **DB\_CALLBACK\_START** The prototype is as follows:

```
void SQL_CALLBACK start_callback( SQLCA * sqlca );
```



This function is called just before a database request is sent to the server. `DB_CALLBACK_START` is used only on Windows.

- **DB\_CALLBACK\_FINISH** The prototype is as follows:

```
void SQL_CALLBACK finish_callback( SQLCA * sqlca );
```

This function is called after the response to a database request has been received by the DBLIB interface DLL. `DB_CALLBACK_FINISH` is used only on Windows operating systems.

- **DB\_CALLBACK\_CONN\_DROPPED** The prototype is as follows:

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

This function is called when the database server is about to drop a connection because of a liveness timeout, through a `DROP CONNECTION` statement, or because the database server is being shut down. The connection name *conn\_name* is passed in to allow you to distinguish between connections. If the connection was not named, it has a value of `NULL`.

- **DB\_CALLBACK\_WAIT** The prototype is as follows:

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

This function is called repeatedly by the interface library while the database server or client library is busy processing your database request.

You would register this callback as follows:

```
db_register_a_callback( &sqlca,
DB_CALLBACK_WAIT,
(SQL_CALLBACK_PARM)&db_wait_request );
```

- **DB\_CALLBACK\_MESSAGE** This is used to enable the application to handle messages received from the server during the processing of a request. Messages can be sent to the client application from the database server using the `SQL MESSAGE` statement. Messages can also be generated by long running database server statements.

The callback prototype is as follows:

```
void SQL_CALLBACK message_callback(
SQLCA * sqlca,
unsigned char msg_type,
an_sql_code code,
unsigned short length,
char * msg
);
```

The *msg\_type* parameter states how important the message is. You may want to handle different message types in different ways. The following possible values for *msg\_type* are defined in *sqldef.h*.

- **MESSAGE\_TYPE\_INFO** The message type was `INFO`.

- **MESSAGE\_TYPE\_WARNING** The message type was WARNING.
- **MESSAGE\_TYPE\_ACTION** The message type was ACTION.
- **MESSAGE\_TYPE\_STATUS** The message type was STATUS.
- **MESSAGE\_TYPE\_PROGRESS** The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

The *code* field may provide a SQLCODE associated with the message, otherwise the value is 0. The *length* field tells you how long the message is. The message is *not* null-terminated. SQL Anywhere DBLIB and ODBC clients can use the DB\_CALLBACK\_MESSAGE parameter to receive progress messages.

For example, the Interactive SQL callback displays STATUS and INFO message on the Messages tab, while messages of type ACTION and WARNING go to a window. If an application does not register this callback, there is a default callback, which causes all messages to be written to the server logfile (if debugging is on and a logfile is specified). In addition, messages of type MESSAGE\_TYPE\_WARNING and MESSAGE\_TYPE\_ACTION are more prominently displayed, in an operating system-dependent manner.

When a message callback is not registered by the application, messages sent to the client are saved to the log file when the LogFile connection parameter is specified. Also, ACTION or STATUS messages sent to the client appear in a window on Windows operating systems and are logged to stderr on Unix operating systems.

- **DB\_CALLBACK\_VALIDATE\_FILE\_TRANSFER** This is used to register a file transfer validation callback function. Before allowing any transfer to take place, the client library will invoke the validation callback, if it exists. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the client library will not allow a transfer unless the client application has registered a validation callback. The conditions under which a validation call is made are described more fully below.

The callback prototype is as follows:

```
int SQL_CALLBACK file_transfer_callback(  
SQLCA * sqlca,  
char * file_name,  
int is_write  
);
```

The *file\_name* parameter is the name of the file to be read or written. The *is\_write* parameter is 0 if a read is requested (transfer from the client to the server), and non-zero for a write. The callback function should return 0 if the file transfer is not allowed, non-zero otherwise.

For data security, the server tracks the origin of statements requesting a file transfer. The server determines if the statement was received directly from the client application. When initiating the transfer of data from the client, the server sends the information about the origin of the statement to the client software. On its part, the embedded SQL client library allows unconditional transfer of data only if the data transfer is being requested due to the execution of a statement sent directly by the

client application. Otherwise, the application must have registered the validation callback described above, in the absence of which the transfer is denied and the statement fails with an error. Note that if the client statement invokes a stored procedure already existing in the database, then the execution of the stored procedure itself is considered not to have been for a client initiated statement. However, if the client application explicitly creates a temporary stored procedure then the execution of the stored procedure results in the server treating the procedure as having been client initiated. Similarly, if the client application executes a batch statement, then the execution of the batch statement is considered as being done directly by the client application.

### See also

- “The SQL Communication Area (SQLCA)” on page 461
- “LogFile (LOG) connection parameter” [*SQL Anywhere Server - Database Administration*]
- “MESSAGE statement” [*SQL Anywhere Server - SQL Reference*]
- “progress\_messages option” [*SQL Anywhere Server - Database Administration*]

## db\_start\_database function

### Syntax

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

### Returns

Non-zero if successful; 0 otherwise.

### Remarks

The database is started on an existing server, if possible. Otherwise, a new server is started.

If the database was already running or was successfully started, the return value is true (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

If a user ID and password are supplied in the parameters, they are ignored.

The permission required to start and stop a database is set on the server command line using the -gd option.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“Locating a database server” \[SQL Anywhere Server - Database Administration\]](#)
- [“-gd dbeng12/dbsrv12 server option” \[SQL Anywhere Server - Database Administration\]](#)

## db\_start\_engine function

**Syntax**

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

**Returns**

Non-zero if successful; 0 otherwise.

**Remarks**

Starts the database server if it is not running.

If the database server was already running or was successfully started, the return value is TRUE (non-zero) and SQLCODE is set to 0. Error information is returned in the SQLCA.

The following call to db\_start\_engine starts the database server, loads the specified database, and names the server demo.

```
db_start_engine( &sqlca, "DBF=demo.db;START=dbeng12" );
```

Unless the ForceStart (FORCE) connection parameter is used and set to YES, the db\_start\_engine function attempts to connect to a server before starting one, to avoid attempting to start a server that is already running.

When the ForceStart connection is set to YES, there is no attempt to connect to a server before trying to start one. This enables the following pair of commands to work as expected:

1. Start a database server named server\_1:

```
dbeng12 -n server_1 demo.db
```

2. Force a new server to start and connect to it:

```
db_start_engine( &sqlca,  
"START=dbeng12 -n server_2 mydb.db;ForceStart=YES" )
```

If ForceStart (FORCE) is not used and the ServerName (Server) parameter is not used, then the second command would have attempted to connect to server\_1. The db\_start\_engine function does not pick up the server name from the -n option of the StartLine (START) parameter.

**See also**

- “The SQL Communication Area (SQLCA)” on page 461
- “Connection parameters” [*SQL Anywhere Server - Database Administration*]
- “Locating a database server” [*SQL Anywhere Server - Database Administration*]

## db\_stop\_database function

**Syntax**

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

**Returns**

Non-zero if successful; 0 otherwise.

**Remarks**

Stop the database identified by DatabaseName (DBN) on the server identified by ServerName (Server). If ServerName is not specified, the default server is used.

By default, this function does not stop a database that has existing connections. If Unconditional (UNC) is set to **yes**, the database is stopped regardless of existing connections.

A return value of TRUE indicates that there were no errors.

The permission required to start and stop a database is set on the server command line using the -gd option.

**See also**

- “The SQL Communication Area (SQLCA)” on page 461
- “Connection parameters” [*SQL Anywhere Server - Database Administration*]
- “-gd dbeng12/dbsrv12 server option” [*SQL Anywhere Server - Database Administration*]

## db\_stop\_engine function

### Syntax

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

### Returns

Non-zero if successful; 0 otherwise.

### Remarks

Stops execution of the database server. The steps carried out by this function are:

- Look for a local database server that has a name that matches the ServerName (Server) parameter. If no ServerName is specified, look for the default local database server.
- If no matching server is found, this function returns with success.
- Send a request to the server to tell it to checkpoint and shut down all databases.
- Unload the database server.

By default, this function does not stop a database server that has existing connections. If the Unconditional=yes connection parameter is specified, the database server is stopped regardless of existing connections.

A C program can use this function instead of spawning dbstop. A return value of TRUE indicates that there were no errors.

The use of db\_stop\_engine is subject to the permissions set with the -gk server option.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“-gk dbeng12/dbsrv12 server option” \[SQL Anywhere Server - Database Administration\]](#)

## db\_string\_connect function

### Syntax

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

**Returns**

Non-zero if successful; 0 otherwise.

**Remarks**

Provides extra functionality beyond the embedded SQL CONNECT statement.

The algorithm used by this function is described in the troubleshooting topic.

The return value is TRUE (non-zero) if a connection was successfully established and FALSE (zero) otherwise. Error information for starting the server, starting the database, or connecting is returned in the SQLCA.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“Troubleshooting connections” \[SQL Anywhere Server - Database Administration\]](#)

## db\_string\_disconnect function

**Syntax**

```
unsigned int db_string_disconnect(
    SQLCA * sqlca,
    char * parms );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **parms** A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD=value**. For example:

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

**Returns**

Non-zero if successful; 0 otherwise.

**Remarks**

This function disconnects the connection identified by the ConnectionName parameter. All other parameters are ignored.

If no `ConnectionName` parameter is specified in the string, the unnamed connection is disconnected. This is equivalent to the embedded SQL `DISCONNECT` statement. The return value is `TRUE` if a connection was successfully ended. Error information is returned in the `SQLCA`.

This function shuts down the database if it was started with the `AutoStop=yes` connection parameter and there are no other connections to the database. It also stops the server if it was started with the `AutoStop=yes` parameter and there are no other databases running.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## db\_string\_ping\_server function

### Syntax

```
unsigned int db_string_ping_server(  
SQLCA * sqlca,  
char * connect_string,  
unsigned int connect_to_db );
```

### Parameters

- **sqlca** A pointer to a `SQLCA` structure.
- **connect\_string** The `connect_string` is a normal connection string that may or may not contain server and database information.
- **connect\_to\_db** If `connect_to_db` is non-zero (`TRUE`), then the function attempts to connect to a database on a server. It returns `TRUE` only if the connection string is sufficient to connect to the named database on the named server.

If `connect_to_db` is zero, then the function only attempts to locate a server. It returns `TRUE` only if the connection string is sufficient to locate a server. It makes no attempt to connect to the database.

### Returns

`TRUE` (non-zero) if the server or database was successfully located; `FALSE` (zero) otherwise. Error information for locating the server or database is returned in the `SQLCA`.

### Remarks

This function can be used to determine if a server can be located, and optionally, if it a successful connection to a database can be made.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)



## db\_time\_change function

### Syntax

```
unsigned int db_time_change(  
SQLCA * sqlca);
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.

### Returns

TRUE if successful; FALSE otherwise.

### Remarks

This function permits clients to notify the server that the time has changed on the client. This function recalculates the time zone adjustment and sends it to the server. On Windows platforms, it is recommended that applications call this function when they receive the WM\_TIMECHANGE message. This will make sure that UTC timestamps are consistent over time changes, time zone changes, or daylight savings time changeovers.

### See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## fill\_s\_sqllda function

### Syntax

```
struct sqllda * fill_s_sqllda(  
struct sqllda * sqllda,  
unsigned int maxlen );
```

### Parameters

- **sqllda** A pointer to a SQLDA structure.
- **maxlen** The maximum number of bytes to allocate for the string.

### Returns

*sqllda* if successful and returns NULL if there is not enough memory available.

### Remarks

The same as fill\_sqllda, except that it changes all the data types in *sqllda* to type DT\_STRING. Enough space is allocated to hold the string representation of the type originally specified by the SQLDA, up to a maximum of *maxlen* bytes. The length fields in the SQLDA (sqlllen) are modified appropriately.

The SQLDA should be freed using the free\_filled\_sqllda function.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## fill\_sqlda function

**Syntax**

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

**Parameters**

- **sqlda** A pointer to a SQLDA structure.

**Returns**

*sqlda* if successful and returns NULL if there is not enough memory available.

**Remarks**

Allocates space for each variable described in each descriptor of *sqlda*, and assigns the address of this memory to the *sqldata* field of the corresponding descriptor. Enough space is allocated for the database type and length indicated in the descriptor.

The SQLDA should be freed using the *free\_filled\_sqlda* function.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“fill\\_sqlda\\_ex function” on page 520](#)
- [“free\\_filled\\_sqlda function” on page 521](#)

## fill\_sqlda\_ex function

**Syntax**

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int flags);
```

**Parameters**

- **sqlda** A pointer to a SQLDA structure.
- **flags** 0 or FILL\_SQLDA\_FLAG\_RETURN\_DT\_LONG

**Returns**

*sqlda* if successful and returns NULL if there is not enough memory available.

**Remarks**

Allocates space for each variable described in each descriptor of *sqlda*, and assigns the address of this memory to the *sqldata* field of the corresponding descriptor. Enough space is allocated for the database type and length indicated in the descriptor.

The SQLDA should be freed using the `free_filled_sqlda` function.

One flag bit is supported: `FILL_SQLDA_FLAG_RETURN_DT_LONG`. This flag is defined in `sqlca.h`.

`FILL_SQLDA_FLAG_RETURN_DT_LONG` preserves `DT_LONGVARCHAR`, `DT_LONGNVARCHAR` and `DT_LONGBINARY` types in the filled descriptor. If this flag bit is not specified, `fill_sqlda_ex` converts `DT_LONGVARCHAR`, `DT_LONGNVARCHAR` and `DT_LONGBINARY` types to `DT_VARCHAR`, `DT_NVARCHAR` and `DT_BINARY` respectively. Using `DT_LONGxyz` types makes it possible to fetch 32767 bytes, not the 32765 bytes that `DT_VARCHAR`, `DT_NVARCHAR` and `DT_BINARY` are limited to.

`fill_sqlda( sqlda )` is equivalent to `fill_sqlda_ex( sqlda, 0 )`.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)
- [“fill\\_sqlda function” on page 520](#)
- [“free\\_filled\\_sqlda function” on page 521](#)

## free\_filled\_sqlda function

**Syntax**

```
void free_filled_sqlda( struct sqlda * sqlda );
```

**Parameters**

- **sqlda** A pointer to a SQLDA structure.

**Remarks**

Free the memory allocated to each *sqldata* pointer and the space allocated for the SQLDA itself. Any null pointer is not freed.

This should only be called if `fill_sqlda`, `fill_sqlda_ex`, or `fill_s_sqlda` was used to allocate the *sqldata* fields of the SQLDA.

Calling this function causes `free_sqlda` to be called automatically, and so any descriptors allocated by `alloc_sqlda` are freed.

### See also

- [“The SQL Communication Area \(SQLCA\)”](#) on page 461
- [“alloc\\_sqllda function”](#) on page 495
- [“fill\\_s\\_sqllda function”](#) on page 519
- [“fill\\_sqllda function”](#) on page 520
- [“fill\\_sqllda\\_ex function”](#) on page 520
- [“free\\_sqllda function”](#) on page 522

## free\_sqllda function

### Syntax

```
void free_sqllda( struct sqllda * sqllda );
```

### Parameters

- **sqllda** A pointer to a SQLDA structure.

### Remarks

Free space allocated to this *sqllda* and free the indicator variable space, as allocated in *fill\_sqllda*. Do not free the memory referenced by each *sqllda* pointer.

### See also

- [“The SQL descriptor area \(SQLDA\)”](#) on page 469
- [“alloc\\_sqllda function”](#) on page 495
- [“fill\\_s\\_sqllda function”](#) on page 519
- [“fill\\_sqllda function”](#) on page 520
- [“fill\\_sqllda\\_ex function”](#) on page 520

## free\_sqllda\_noind function

### Syntax

```
void free_sqllda_noind( struct sqllda * sqllda );
```

### Parameters

- **sqllda** A pointer to a SQLDA structure.

### Remarks

Free space allocated to this *sqllda*. Do not free the memory referenced by each *sqllda* pointer. The indicator variable pointers are ignored.

**See also**

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)
- [“alloc\\_sqlda function” on page 495](#)
- [“fill\\_s\\_sqlda function” on page 519](#)
- [“fill\\_sqlda function” on page 520](#)
- [“fill\\_sqlda\\_ex function” on page 520](#)

## sql\_needs\_quotes function

**Syntax**

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

**Parameters**

- **sqlca** A pointer to a SQLCA structure.
- **str** A string of characters that is a candidate for a SQL identifier.

**Returns**

TRUE or FALSE indicating whether the string requires double quotes around it when it is used as a SQL identifier.

**Remarks**

This function formulates a request to the database server to determine if quotes are needed. Relevant information is stored in the sqlcode field.

There are three cases of return value/code combinations:

- **return = FALSE, sqlcode = 0** The string does not need quotes.
- **return = TRUE** The sqlcode is always SQLE\_WARNING, and the string requires quotes.
- **return = FALSE** If sqlcode is something other than 0 or SQLE\_WARNING, the test is inconclusive.

**See also**

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

## sqlda\_storage function

**Syntax**

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

**Parameters**

- **sqlda** A pointer to a SQLDA structure.

- **varno** An index for a sqlvar host variable.

### Returns

An unsigned 32-bit integer value representing the amount of storage required to store any value for the variable described in `sqlda->sqlvar[varno]`.

### See also

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)

## sqlda\_string\_length function

### Syntax

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int varno );
```

### Parameters

- **sqlda** A pointer to a SQLDA structure.
- **varno** An index for a sqlvar host variable.

### Returns

An unsigned 32-bit integer value representing the length of the C string (type DT\_STRING) that would be required to hold the variable `sqlda->sqlvar[varno]` (no matter what its type is).

### See also

- [“The SQL descriptor area \(SQLDA\)” on page 469](#)

## sqlerror\_message function

### Syntax

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

### Parameters

- **sqlca** A pointer to a SQLCA structure.
- **buffer** The buffer in which to place the message (up to *max* characters).
- **max** The maximum length of the buffer.

### Returns

A pointer to a string that contains an error message or NULL if no error was indicated.

## Remarks

Returns a pointer to a string that contains an error message. The error message contains text for the error code in the SQLCA. If no error was indicated, a null pointer is returned. The error message is placed in the buffer supplied, truncated to length *max* if necessary.

## See also

- [“The SQL Communication Area \(SQLCA\)” on page 461](#)

# Embedded SQL statement summary

ALL embedded SQL statements must be preceded with EXEC SQL and end with a semicolon (;).

There are two groups of embedded SQL statements. Standard SQL statements are used by simply placing them in a C program enclosed with EXEC SQL and a semicolon (;). CONNECT, DELETE, SELECT, SET, and UPDATE have additional formats only available in embedded SQL. The additional formats fall into the second category of embedded SQL specific statements.

For descriptions of the standard SQL statements, see [“SQL statements” \[SQL Anywhere Server - SQL Reference\]](#).

Several SQL statements are specific to embedded SQL and can only be used in a C program. See [“SQL language elements” \[SQL Anywhere Server - SQL Reference\]](#).

Standard data manipulation and data definition statements can be used from embedded SQL applications. In addition, the following statements are specifically for embedded SQL programming:

- [“ALLOCATE DESCRIPTOR statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#) allocate memory for a descriptor.
- [“CLOSE statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#) close a cursor.
- [“CONNECT statement \[ESQL\] \[Interactive SQL\]” \[SQL Anywhere Server - SQL Reference\]](#) connect to the database.
- [“DEALLOCATE DESCRIPTOR statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#) reclaim memory for a descriptor.
- [“Declaration section \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#) declare host variables for database communication.
- [“DECLARE CURSOR statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#) declare a cursor.
- [“DELETE statement \(positioned\) \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#) delete the row at the current position in a cursor.
- [“DESCRIBE statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#) describe the host variables for a particular SQL statement.

- **“DISCONNECT statement [ESQL] [Interactive SQL]” [SQL Anywhere Server - SQL Reference]** disconnect from database server.
- **“DROP STATEMENT statement [ESQL]” [SQL Anywhere Server - SQL Reference]** free resources used by a prepared statement.
- **“EXECUTE statement [ESQL]” [SQL Anywhere Server - SQL Reference]** execute a particular SQL statement.
- **“EXPLAIN statement [ESQL]” [SQL Anywhere Server - SQL Reference]** explain the optimization strategy for a particular cursor.
- **“FETCH statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference]** fetch a row from a cursor.
- **“GET DATA statement [ESQL]” [SQL Anywhere Server - SQL Reference]** fetch long values from a cursor.
- **“GET DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference]** retrieve information about a variable in a SQLDA.
- **“GET OPTION statement [ESQL]” [SQL Anywhere Server - SQL Reference]** get the setting for a particular database option.
- **“INCLUDE statement [ESQL]” [SQL Anywhere Server - SQL Reference]** include a file for SQL preprocessing.
- **“OPEN statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference]** open a cursor.
- **“PREPARE statement [ESQL]” [SQL Anywhere Server - SQL Reference]** prepare a particular SQL statement.
- **“PUT statement [ESQL]” [SQL Anywhere Server - SQL Reference]** insert a row into a cursor.
- **“SET CONNECTION statement [Interactive SQL] [ESQL]” [SQL Anywhere Server - SQL Reference]** change active connection.
- **“SET DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference]** describe the variables in a SQLDA and place data into the SQLDA.
- **“SET SQLCA statement [ESQL]” [SQL Anywhere Server - SQL Reference]** use a SQLCA other than the default global one.
- **“UPDATE (positioned) statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference]** update the row at the current location of a cursor.
- **“WHENEVER statement [ESQL]” [SQL Anywhere Server - SQL Reference]** specify actions to occur on errors in SQL statements.



---

# SQL Anywhere database API for C/C++

The SQL Anywhere C application programming interface (API) is a data access API for the C / C++ languages. The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using the SQL Anywhere C API, your C / C++ applications have direct access to SQL Anywhere database servers.

## SQL Anywhere C API support

The SQL Anywhere C application programming interface (API) is a data access API for the C / C++ languages. The C API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using the SQL Anywhere C API, your C / C++ applications have direct access to SQL Anywhere database servers.

The SQL Anywhere C API simplifies the creation of C and C++ wrapper drivers for several interpreted programming languages including PHP, Perl, Python, and Ruby. The SQL Anywhere C API is layered on top of the DBLIB package and it was implemented with Embedded SQL.

Although it is not a replacement for DBLIB, the SQL Anywhere C API simplifies the creation of applications using C and C++. You do not need an advanced knowledge of Embedded SQL to use the SQL Anywhere C API.

### API distribution

The API is built as a dynamic link library (DLL) (*dbcapi.dll*) on Microsoft Windows systems and as a shared object (*libdbcapi.so*) on Unix systems. The DLL is statically linked to the DBLIB package of the SQL Anywhere version on which it is built. When the *dbcapi.dll* file is loaded, the corresponding *dblibX.dll* file is loaded by the operating system. Applications using *dbcapi.dll* can either link directly to it or load it dynamically.

Descriptions of the SQL Anywhere C API data types and entry points are provided in the *sacapi.h* header file which is located in the *sdk\dbcapi* directory of your SQL Anywhere installation.

### Threading support

The SQL Anywhere C API library is thread-unaware; the library does not perform any tasks that require mutual exclusion. To allow the library to work in threaded applications, only one request is allowed on a single connection. With this rule, the application is responsible for doing mutual exclusion when accessing any connection-specific resource. This includes connection handles, prepared statements, and result set objects.

### Examples

Examples that show how to use the SQL Anywhere C API can be found in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

- **connecting.cpp** This is an example of how to create a connection object and connect with it to SQL Anywhere.

- ***dbcapi\_isql.cpp*** This example shows how to write an ISQL-like application.
- ***fetching\_a\_result\_set.cpp*** This example shows how to fetch data from a result set.
- ***fetching\_multiple\_from\_sp.cpp*** This example shows how to fetch multiple result sets from a stored procedure.
- ***preparing\_statements.cpp*** This example shows how to prepare and execute a statement.
- ***send\_retrieve\_full\_blob.cpp*** This example shows how to insert and retrieve a blob in one chunk.
- ***send\_retrieve\_part\_blob.cpp*** This example shows how to insert a blob in chunks and how to retrieve it in chunks as well.

## Dynamic loading of the interface library

The code to dynamically load the DLL is contained in the *sacapidll.c* source file which is located in the *sdk\dbcapi* directory of your SQL Anywhere installation. Applications must use the *sacapidll.h* header file and include the source code in *sacapidll.c*. You can use the `sqlany_initialize_interface` method to dynamically load the DLL and look up the entry points. Examples are provided with the SQL Anywhere installation.

## SQL Anywhere C API reference

### Header files

- `sacapi.h`
- `sacapidll.h`

### Remarks

The *sacapi.h* header file defines the SQL Anywhere C API entry points.

The *sacapidll.h* header file defines the C API library initialization and finalization functions. You must include *sacapidll.h* in your source files and include the source code from *sacapidll.c*.

## sqlany\_affected\_rows method

Returns the number of rows affected by execution of the prepared statement.

### Syntax

```
public sacapi_i32 sqlany_affected_rows(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement that was prepared and executed successfully with no result set returned. For example, an INSERT, UPDATE or DELETE statement was executed.

### Returns

The number of rows affected or -1 on failure.

### See also

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)

## sqlany\_bind\_param method

Bind a user-supplied buffer as a parameter to the prepared statement.

### Syntax

```
public sacapi_bool sqlany_bind_param(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param * param
)
```

### Parameters

- **sqlany\_stmt** A statement prepared successfully using sqlany\_prepare().
- **index** The index of the parameter. This number must be between 0 and sqlany\_num\_params() - 1.
- **param** A a\_sqlany\_bind\_param structure description of the parameter to be bound.

### Returns

1 on success or 0 on unsuccessful.

### See also

- [“sqlany\\_describe\\_bind\\_param method \[SQL Anywhere C\]” on page 532](#)

## sqlany\_cancel method

Cancel an outstanding request on a connection.

### Syntax

```
public void sqlany_cancel(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** A connection object with a connection established using sqlany\_connect().

## sqlany\_clear\_error method

Clears the last stored error code.

### Syntax

```
public void sqlany_clear_error(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** A connection object returned from `sqlany_new_connection()`.

### See also

- [“sqlany\\_new\\_connection method \[SQL Anywhere C\]” on page 546](#)

## sqlany\_client\_version method

Returns the current client version.

### Syntax

```
public sacapi_bool sqlany_client_version(char * buffer, size_t len)
```

### Parameters

- **buffer** The buffer to be filled with the client version string.
- **len** The length of the buffer supplied.

### Returns

1 when successful or 0 when unsuccessful.

### Remarks

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

## sqlany\_client\_version\_ex method

Returns the current client version.

### Syntax

```
public sacapi_bool sqlany_client_version_ex(  
    a_sqlany_interface_context * context,  
    char * buffer,  
    size_t len  
)
```

### Parameters

- **context** object that was create with `sqlany_init_ex()`
- **buffer** The buffer to be filled with the client version string.
- **len** The length of the buffer supplied.

### Returns

1 when successful or 0 when unsuccessful.

### Remarks

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

### See also

- [“sqlany\\_init\\_ex method \[SQL Anywhere C\]” on page 544](#)

## sqlany\_commit method

Commits the current transaction.

### Syntax

```
public sacapi_bool sqlany_commit(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** The connection object on which the commit operation is performed.

### Returns

1 when successful or 0 when unsuccessful.

### See also

- [“sqlany\\_rollback method \[SQL Anywhere C\]” on page 550](#)

## sqlany\_connect method

Creates a connection to a SQL Anywhere database server using the supplied connection object and connection string.

### Syntax

```
public sacapi_bool sqlany_connect(  
    a_sqlany_connection * sqlany_conn,  
    const char * str  
)
```

**Parameters**

- **sqlany\_conn** A connection object created by `sqlany_new_connection()`.
- **str** A SQL Anywhere connection string.

**Returns**

1 if the connection is established successfully or 0 when the connection fails. Use `sqlany_error()` to retrieve the error code and message.

**Remarks**

The supplied connection object must first be allocated using `sqlany_new_connection()`.

The following example demonstrates how to retrieve the error code of a failed connection attempt:

```
a_sqlany_connection * sqlany_conn;
sqlany_conn = sqlany_new_connection();
if( !sqlany_connect( sqlany_conn, "uid=dba;pwd=sql" ) ) {
    char reason[SACAPI_ERROR_SIZE];
    sacapi_i32 code;
    code = sqlany_error( sqlany_conn, reason, sizeof(reason) );
    printf( "Connection failed. Code: %d Reason: %s\n", code, reason );
} else {
    printf( "Connected successfully!\n" );
    sqlany_disconnect( sqlany_conn );
}
sqlany_free_connection( sqlany_conn );
```

For more information on connecting to a SQL Anywhere database server, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#) and [“SQL Anywhere database connections” \[SQL Anywhere Server - Database Administration\]](#).

**See also**

- [“sqlany\\_new\\_connection method \[SQL Anywhere C\]” on page 546](#)
- [“sqlany\\_error method \[SQL Anywhere C\]” on page 533](#)

## sqlany\_describe\_bind\_param method

Describes the bind parameters of a prepared statement.

**Syntax**

```
public sacapi_bool sqlany_describe_bind_param(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param * param
)
```

**Parameters**

- **sqlany\_stmt** A statement prepared successfully using `sqlany_prepare()`.
- **index** The index of the parameter. This number must be between 0 and `sqlany_num_params() - 1`.

- **param** A `a_sqlany_bind_param` structure that is populated with information.

### Returns

1 when successful or 0 when unsuccessful.

### Remarks

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement, stored procedure or a DML, determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

### See also

- [“sqlany\\_bind\\_param method \[SQL Anywhere C\]” on page 529](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_disconnect method

Disconnects an already established SQL Anywhere connection.

### Syntax

```
public sacapi_bool sqlany_disconnect(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** A connection object with a connection established using `sqlany_connect()`.

### Returns

1 when successful or 0 when unsuccessful.

### Remarks

All uncommitted transactions are rolled back.

### See also

- [“sqlany\\_connect method \[SQL Anywhere C\]” on page 531](#)
- [“sqlany\\_new\\_connection method \[SQL Anywhere C\]” on page 546](#)

## sqlany\_error method

Retrieves the last error code and message stored in the connection object.

### Syntax

```
public sacapi_i32 sqlany_error(  
    a_sqlany_connection * sqlany_conn,  
    char * buffer,
```

```
        size_t size
    )
```

### Parameters

- **sqlany\_conn** A connection object returned from `sqlany_new_connection()`.
- **buffer** A buffer to be filled with the error message.
- **size** The size of the supplied buffer.

### Returns

The last error code. Positive values are warnings, negative values are errors, and 0 indicates success.

### Remarks

For more information on SQLCODE error messages, see [“SQL Anywhere error messages sorted by SQLCODE” \[Error Messages\]](#).

### See also

- [“sqlany\\_connect method \[SQL Anywhere C\]” on page 531](#)

## sqlany\_execute method

Executes a prepared statement.

### Syntax

```
public sacapi_bool sqlany_execute(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement prepared successfully using `sqlany_prepare()`.

### Returns

1 if the statement is executed successfully or 0 on failure.

### Remarks

You can use `sqlany_num_cols()` to verify if the executed statement returned a result set.

The following example shows how to execute a statement that does not return a result set:

```
a_sqlany_stmt *      stmt;
int                 i;
a_sqlany_bind_param param;

stmt = sqlany_prepare( sqlany_conn, "insert into moe(id,value)
values( ?,? )" );
if( stmt ) {
    sqlany_describe_bind_param( stmt, 0, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
}
```



```

    sqlany_bind_param( stmt, 0, &param );

    sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 1, &param );

    for( i = 0; i < 10; i++ ) {
        if( !sqlany_execute( stmt ) ) {
            // call sqlany_error()
        }
    }
    sqlany_free_stmt( stmt );
}

```

**See also**

- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_execute\_direct method

Executes the SQL statement specified by the string argument and possibly returns a result set.

**Syntax**

```

public a_sqlany_stmt * sqlany_execute_direct(
    a_sqlany_connection * sqlany_conn,
    const char * sql_str
)

```

**Parameters**

- **sqlany\_conn** A connection object with a connection established using `sqlany_connect()`.
- **sql\_str** A SQL string. The SQL string should not have parameters such as `?`.

**Returns**

A statement handle if the function executes successfully, NULL when the function executes unsuccessfully.

**Remarks**

Use this method if you want to prepare and execute a statement, or instead of calling `sqlany_prepare()` followed by `sqlany_execute()`.

The following example shows how to execute a statement that returns a result set:

```

a_sqlany_stmt * stmt;

stmt = sqlany_execute_direct( sqlany_conn, "select * from employees" );
if( stmt && sqlany_num_cols( stmt ) > 0 ) {
    while( sqlany_fetch_next( stmt ) ) {
        int i;
        for( i = 0; i < sqlany_num_cols( stmt ); i++ ) {
            // Get column i data
        }
    }
}

```

```
    }  
    sqlany_free_stmt( stmt );  
}
```

**Note**

This function cannot be used for executing a SQL statement with parameters.

**See also**

- [“sqlany\\_fetch\\_absolute method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_fetch\\_next method \[SQL Anywhere C\]” on page 537](#)
- [“sqlany\\_num\\_cols method \[SQL Anywhere C\]” on page 547](#)
- [“sqlany\\_get\\_column method \[SQL Anywhere C\]” on page 540](#)

## sqlany\_execute\_immediate method

Executes the supplied SQL statement immediately without returning a result set.

**Syntax**

```
public sacapi_bool sqlany_execute_immediate(  
    a_sqlany_connection * sqlany_conn,  
    const char * sql  
)
```

**Parameters**

- **sqlany\_conn** A connection object with a connection established using `sqlany_connect()`.
- **sql** A string representing the SQL statement to be executed.

**Returns**

1 on success or 0 on failure.

**Remarks**

This function is useful for SQL statements that do not return a result set.

## sqlany\_fetch\_absolute method

Moves the current row in the result set to the row number specified and then fetches the data at that row.

**Syntax**

```
public sacapi_bool sqlany_fetch_absolute(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_i32 row_num  
)
```

### Parameters

- **sqlany\_stmt** A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **row\_num** The row number to be fetched. The first row is 1, the last row is -1.

### Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

### See also

- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_error method \[SQL Anywhere C\]” on page 533](#)
- [“sqlany\\_fetch\\_next method \[SQL Anywhere C\]” on page 537](#)

## sqlany\_fetch\_next method

Returns the next row from the result set.

### Syntax

```
public sacapi_bool sqlany_fetch_next(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.

### Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

### Remarks

This function fetches the next row from the result set. When the result object is first created, the current row pointer is set to before the first row, that is, row 0. This function first advances the row pointer and then fetches the data at the new row.

### See also

- [“sqlany\\_fetch\\_absolute method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_error method \[SQL Anywhere C\]” on page 533](#)

## sqlany\_finalize\_interface method

Unloads the C API DLL library and resets the `SQLAnywhereInterface` structure.

### Syntax

```
public void sqlany_finalize_interface(SQLAnywhereInterface * api)
```

### Parameters

- **api** An initialized structure to finalize.

### Remarks

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

Use this method to finalize and free resources associated with the SQL Anywhere C API DLL.

Examples of how the `sqlany_finalize_interface` method is used can be found in the C API examples in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

## sqlany\_fini method

Finalizes the interface.

### Syntax

```
public void sqlany_fini()
```

### Remarks

Frees any resources allocated by the API.

### See also

- [“sqlany\\_init method \[SQL Anywhere C\]” on page 543](#)

## sqlany\_fini\_ex method

Finalize the interface that was created using the specified context.

### Syntax

```
public void sqlany_fini_ex(a_sqlany_interface_context * context)
```

### Parameters

- **context** A context object that was returned from `sqlany_init_ex()`

### See also

- [“sqlany\\_init\\_ex method \[SQL Anywhere C\]” on page 544](#)

## sqlany\_free\_connection method

Frees the resources associated with a connection object.

### Syntax

```
public void sqlany_free_connection(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** A connection object created with `sqlany_new_connection()`.

### See also

- [“sqlany\\_new\\_connection method \[SQL Anywhere C\]” on page 546](#)

## sqlany\_free\_stmt method

Frees resources associated with a prepared statement object.

### Syntax

```
public void sqlany_free_stmt(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement object returned by the successful execution of `sqlany_prepare()` or `sqlany_execute_direct()`.

### See also

- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)

## sqlany\_get\_bind\_param\_info method

Retrieves information about the parameters that were bound using `sqlany_bind_param()`.

### Syntax

```
public sacapi_bool sqlany_get_bind_param_info(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 index,  
    a_sqlany_bind_param_info * info  
)
```

### Parameters

- **sqlany\_stmt** A statement prepared successfully using `sqlany_prepare()`.
- **index** The index of the parameter. This number should be between 0 and `sqlany_num_params() - 1`.

- **info** A `sqlany_bind_param_info` buffer to be populated with the bound parameter's information.

### Returns

1 on success or 0 on failure.

### See also

- [“sqlany\\_bind\\_param method \[SQL Anywhere C\]” on page 529](#)
- [“sqlany\\_describe\\_bind\\_param method \[SQL Anywhere C\]” on page 532](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_get\_column method

Fills the supplied buffer with the value fetched for the specified column.

### Syntax

```
public sacapi_bool sqlany_get_column(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 col_index,  
    a_sqlany_data_value * buffer  
)
```

### Parameters

- **sqlany\_stmt** A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **col\_index** The number of the column to be retrieved. A column number is between 0 and `sqlany_num_cols() - 1`.
- **buffer** A `a_sqlany_data_value` object to be filled with the data fetched for column `col_index`.

### Returns

1 on success or 0 for failure. A failure can happen if any of the parameters are invalid or if there is not enough memory to retrieve the full value from the SQL Anywhere database server.

### Remarks

For `A_BINARY` and `A_STRING` \* data types, `value->buffer` points to an internal buffer associated with the result set. Do not rely upon or alter the content of the pointer `buffer` as it changes when a new row is fetched or when the result set object is freed. Users should copy the data out of those pointers into their own buffers.

The `value->length` field indicates the number of valid characters that `value->buffer` points to. The data returned in `value->buffer` is not null-terminated. This function fetches all the returned values from the SQL Anywhere database server. For example, if the column contains a 2GB blob, this function attempts to allocate enough memory to hold that value. If you do not want to allocate memory, use `sqlany_get_data()` instead.

**See also**

- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_fetch\\_absolute method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_fetch\\_next method \[SQL Anywhere C\]” on page 537](#)

## sqlany\_get\_column\_info method

Retrieves column metadata information and fills the `a_sqlany_column_info` structure with information about the column.

**Syntax**

```
public sacapi_bool sqlany_get_column_info(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_column_info * buffer
)
```

**Parameters**

- **sqlany\_stmt** A statement object created by `sqlany_prepare()` or `sqlany_execute_direct()`.
- **col\_index** The column number between 0 and `sqlany_num_cols() - 1`.
- **buffer** A column info structure to be filled with column information.

**Returns**

1 on success or 0 if the column index is out of range, or if the statement does not return a result set.

**See also**

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_get\_data method

Retrieves the data fetched for the specified column into the supplied buffer memory.

**Syntax**

```
public sacapi_i32 sqlany_get_data(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    size_t offset,
    void * buffer,
    size_t size
)
```

**Parameters**

- **sqlany\_stmt** A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **col\_index** The number of the column to be retrieved. A column number is between 0 and `sqlany_num_cols() - 1`.
- **offset** The starting offset of the data to get.
- **buffer** A buffer to be filled with the contents of the column. The buffer pointer must be aligned correctly for the data type copied into it.
- **size** The size of the buffer in bytes. The function fails if you specify a size greater than 2GB.

**Returns**

The number of bytes successfully copied into the supplied buffer. This number must not exceed 2GB. 0 indicates that no data remains to be copied. -1 indicates a failure.

**See also**

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_fetch\\_absolute method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_fetch\\_next method \[SQL Anywhere C\]” on page 537](#)

## sqlany\_get\_data\_info method

Retrieves information about the data that was fetched by the last fetch operation.

**Syntax**

```
public sacapi_bool sqlany_get_data_info(  
    a_sqlany_stmt * sqlany_stmt,  
    sacapi_u32 col_index,  
    a_sqlany_data_info * buffer  
)
```

**Parameters**

- **sqlany\_stmt** A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.
- **col\_index** The column number between 0 and `sqlany_num_cols() - 1`.
- **buffer** A data info buffer to be filled with the metadata about the data fetched.

**Returns**

1 on success, and 0 on failure. Failure is returned when any of the supplied parameters are invalid.



**See also**

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_fetch\\_absolute method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_fetch\\_next method \[SQL Anywhere C\]” on page 537](#)

## sqlany\_get\_next\_result method

Advances to the next result set in a multiple result set query.

**Syntax**

```
public sacapi_bool sqlany_get_next_result(a_sqlany_stmt * sqlany_stmt)
```

**Parameters**

- **sqlany\_stmt** A statement object executed by `sqlany_execute()` or `sqlany_execute_direct()`.

**Returns**

1 if the statement successfully advances to the next result set, 0 otherwise.

**Remarks**

If a query (such as a call to a stored procedure) returns multiple result sets, then this function advances from the current result set to the next.

The following example demonstrates how to advance to the next result set in a multiple result set query:

```
stmt = sqlany_execute_direct( sqlany_conn, "call
my_multiple_results_procedure()" );
if( result ) {
    do {
        while( sqlany_fetch_next( stmt ) ) {
            // get column data
        }
    } while( sqlany_get_next_result( stmt ) );
    sqlany_free_stmt( stmt );
}
```

**See also**

- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)

## sqlany\_init method

Initializes the interface.

**Syntax**

```
public sacapi_bool sqlany_init(
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

**Parameters**

- **app\_name** A string that names the application that is using the API. For example, "PHP", "PERL", or "RUBY".
- **api\_version** The version of the compiled application.
- **version\_available** An optional argument to return the maximum supported API version.

**Returns**

1 on success, 0 otherwise

**Remarks**

The following example demonstrates how to initialize the SQL Anywhere C API DLL:

```
sacapi_u32 api_version;
if( sqlany_init( "PHP", SQLANY_API_VERSION_1, &api_version ) ) {
    printf( "Interface initialized successfully!\n" );
} else {
    printf( "Failed to initialize the interface! Supported version=%d\n",
api_version );
}
```

**See also**

- [“sqlany\\_fini method \[SQL Anywhere C\]” on page 538](#)

## sqlany\_init\_ex method

Initializes the interface using a context.

**Syntax**

```
public a_sqlany_interface_context * sqlany_init_ex(
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

**Parameters**

- **app\_name** A string that names the API used, for example "PHP", "PERL", or "RUBY".
- **api\_version** The current API version that the application is using. This should normally be one of the SQLANY\_API\_VERSION\_\* macros

- **version\_available** An optional argument to return the maximum API version that is supported.

### Returns

a context object on success and NULL on failure.

### See also

- [“sqlany\\_fini\\_ex method \[SQL Anywhere C\]” on page 538](#)

## sqlany\_initialize\_interface method

Initializes the SQLAnywhereInterface object and loads the DLL dynamically.

### Syntax

```
public int sqlany_initialize_interface(  
    SQLAnywhereInterface * api,  
    const char * optional_path_to_dll  
)
```

### Parameters

- **api** An API structure to initialize.
- **optional\_path\_to\_dll** An optional argument that specifies a path to the SQL Anywhere C API DLL.

### Returns

1 on successful initialization, and 0 on failure.

### Remarks

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

This function attempts to load the SQL Anywhere C API DLL dynamically and looks up all the entry points of the DLL. The fields in the SQLAnywhereInterface structure are populated to point to the corresponding functions in the DLL. If the optional path argument is NULL, the environment variable SQLANY\_DLL\_PATH is checked. If the variable is set, the library attempts to load the DLL specified by the environment variable. If that fails, the interface attempts to load the DLL directly (this relies on the environment being setup correctly).

Examples of how the sqlany\_initialize\_interface method is used can be found in the C API examples in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

## sqlany\_make\_connection method

Creates a connection object based on a supplied DBLIB SQLCA pointer.

**Syntax**

```
public a_sqlany_connection * sqlany_make_connection(void * arg)
```

**Parameters**

- **arg** A void \* pointer to a DBLIB SQLCA object.

**Returns**

A connection object.

**See also**

- [“sqlany\\_new\\_connection method \[SQL Anywhere C\]” on page 546](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute\\_immediate method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_make\_connection\_ex method

Creates a connection object based on a supplied DBLIB SQLCA pointer and context.

**Syntax**

```
public a_sqlany_connection * sqlany_make_connection_ex(  
    a_sqlany_interface_context * context,  
    void * arg  
)
```

**Parameters**

- **context** A valid context object that was created by `sqlany_init_ex()`
- **arg** A void \* pointer to a DBLIB SQLCA object.

**Returns**

A connection object.

**See also**

- [“sqlany\\_init\\_ex method \[SQL Anywhere C\]” on page 544](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_execute\\_immediate method \[SQL Anywhere C\]” on page 536](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_new\_connection method

Creates a connection object.

**Syntax**

```
public a_sqlany_connection * sqlany_new_connection(void)
```

**Returns**

A connection object

**Remarks**

You must create an API connection object before establishing a database connection. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time. In addition, not more than one thread is allowed to access a connection object at a time. Undefined behavior or a failure occurs when multiple threads attempt to access a connection object simultaneously.

**See also**

- [“sqlany\\_connect method \[SQL Anywhere C\]” on page 531](#)
- [“sqlany\\_disconnect method \[SQL Anywhere C\]” on page 533](#)

## sqlany\_new\_connection\_ex method

Creates a connection object using a context.

**Syntax**

```
public a_sqlany_connection * sqlany_new_connection_ex(  
    a_sqlany_interface_context * context  
)
```

**Parameters**

- **context** A context object that was returned from `sqlany_init_ex()`

**Returns**

A connection object

**See also**

- [“sqlany\\_connect method \[SQL Anywhere C\]” on page 531](#)
- [“sqlany\\_disconnect method \[SQL Anywhere C\]” on page 533](#)
- [“sqlany\\_init\\_ex method \[SQL Anywhere C\]” on page 544](#)

## sqlany\_num\_cols method

Returns number of columns in the result set.

**Syntax**

```
public sacapi_i32 sqlany_num_cols(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement object created by `sqlany_prepare()` or `sqlany_execute_direct()`.

### Returns

The number of columns in the result set or -1 on a failure.

### See also

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_execute\\_direct method \[SQL Anywhere C\]” on page 535](#)
- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_num\_params method

Returns the number of parameters expected for a prepared statement.

### Syntax

```
public sacapi_i32 sqlany_num_params(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement object returned by the successful execution of `sqlany_prepare()`.

### Returns

The expected number of parameters, or -1 if the statement object is not valid.

### See also

- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_num\_rows method

Returns the number of rows in the result set.

### Syntax

```
public sacapi_i32 sqlany_num_rows(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement object that was executed by `sqlany_execute()` or `sqlany_execute_direct()`.

### Returns

The number rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

## Remarks

By default this function only returns an estimate. To return an exact count, set the `row_counts` option on the connection. For more information on the `row_counts` option, see “[row\\_counts option](#)” [*SQL Anywhere Server - Database Administration*].

## See also

- “[sqlany\\_execute\\_direct method \[SQL Anywhere C\]](#)” on page 535
- “[sqlany\\_execute method \[SQL Anywhere C\]](#)” on page 534

# sqlany\_prepare method

Prepares a supplied SQL string.

## Syntax

```
public a_sqlany_stmt * sqlany_prepare(  
    a_sqlany_connection * sqlany_conn,  
    const char * sql_str  
)
```

## Parameters

- **sqlany\_conn** A connection object with a connection established using `sqlany_connect()`.
- **sql\_str** The SQL statement to be prepared.

## Returns

A handle to a SQL Anywhere statement object. The statement object can be used by `sqlany_execute()` to execute the statement.

## Remarks

Execution does not happen until `sqlany_execute()` is called. The returned statement object should be freed using `sqlany_free_stmt()`.

The following statement demonstrates how to prepare a SELECT SQL string:

```
char * str;  
a_sqlany_stmt * stmt;  
  
str = "select * from employees where salary >= ?";  
stmt = sqlany_prepare( sqlany_conn, str );  
if( stmt == NULL ) {  
    // Failed to prepare statement, call sqlany_error() for more info  
}
```

### See also

- [“sqlany\\_free\\_stmt method \[SQL Anywhere C\]” on page 539](#)
- [“sqlany\\_connect method \[SQL Anywhere C\]” on page 531](#)
- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)
- [“sqlany\\_num\\_params method \[SQL Anywhere C\]” on page 548](#)
- [“sqlany\\_describe\\_bind\\_param method \[SQL Anywhere C\]” on page 532](#)
- [“sqlany\\_bind\\_param method \[SQL Anywhere C\]” on page 529](#)

## sqlany\_reset method

Resets a statement to its prepared state condition.

### Syntax

```
public sacapi_bool sqlany_reset(a_sqlany_stmt * sqlany_stmt)
```

### Parameters

- **sqlany\_stmt** A statement prepared successfully using `sqlany_prepare()`.

### Returns

1 on success, 0 on failure.

### See also

- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_rollback method

Rolls back the current transaction.

### Syntax

```
public sacapi_bool sqlany_rollback(a_sqlany_connection * sqlany_conn)
```

### Parameters

- **sqlany\_conn** The connection object on which the rollback operation is to be performed.

### Returns

1 on success, 0 otherwise.

### See also

- [“sqlany\\_commit method \[SQL Anywhere C\]” on page 531](#)



## sqlany\_send\_param\_data method

Sends data as part of a bound parameter.

### Syntax

```
public sacapi_bool sqlany_send_param_data(
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    char * buffer,
    size_t size
)
```

### Parameters

- **sqlany\_stmt** A statement prepared successfully using `sqlany_prepare()`.
- **index** The index of the parameter. This should be a number between 0 and `sqlany_num_params() - 1`.
- **buffer** The data to be sent.
- **size** The number of bytes to send.

### Returns

1 on success or 0 on failure.

### Remarks

This method can be used to send a large amount of data for a bound parameter in chunks.

### See also

- [“sqlany\\_prepare method \[SQL Anywhere C\]” on page 549](#)

## sqlany\_sqlstate method

Retrieves the current SQLSTATE.

### Syntax

```
public size_t sqlany_sqlstate(
    a_sqlany_connection * sqlany_conn,
    char * buffer,
    size_t size
)
```

### Parameters

- **sqlany\_conn** A connection object returned from `sqlany_new_connection()`.
- **buffer** A buffer to be filled with the current 5-character SQLSTATE.

- **size** The buffer size.

**Returns**

The number of bytes copied into the buffer.

**Remarks**

For more information on SQLSTATE error messages, see “[SQL Anywhere error messages sorted by SQLSTATE](#)” [*Error Messages*].

**See also**

- “[sqlany\\_error method \[SQL Anywhere C\]”](#) on page 533

## a\_sqlany\_data\_direction enumeration

A data direction enumeration.

**Syntax**

```
public enum a_sqlany_data_direction
```

**Members**

Member name	Description	Value
DD_INVALID	Invalid data direction.	0x0
DD_INPUT	Input-only host variables.	0x1
DD_OUTPUT	Output-only host variables.	0x2
DD_INPUT_OUTPUT	Input and output host variables.	0x3

## a\_sqlany\_data\_type enumeration

Specifies the data type being passed in or retrieved.

**Syntax**

```
public enum a_sqlany_data_type
```

**Members**

Member name	Description
A_INVALID_TYPE	Invalid data type.

Member name	Description
A_BINARY	Binary data. Binary data is treated as-is and no character set conversion is performed.
A_STRING	String data. The data where character set conversion is performed.
A_DOUBLE	Double data. Includes float values.
A_VAL64	64-bit integer.
A_UVAL64	64-bit unsigned integer.
A_VAL32	32-bit integer.
A_UVAL32	32-bit unsigned integer.
A_VAL16	16-bit integer.
A_UVAL16	16-bit unsigned integer.
A_VAL8	8-bit integer.
A_UVAL8	8-bit unsigned integer.

## a\_sqlany\_native\_type enumeration

An enumeration of the native types of values as described by the server.

### Syntax

```
public enum a_sqlany_native_type
```

### Members

Member name	Description
DT_NOTYPE	No data type.
DT_DATE	Null-terminated character string that is a valid date.
DT_TIME	Null-terminated character string that is a valid time.
DT_TIMESTAMP	Null-terminated character string that is a valid timestamp.

Member name	Description
DT_VARCHAR	Varying length character string, in the CHAR character set, with a two-byte length field. The maximum length is 32765 bytes. When sending data, you must set the length field. When fetching data, the database server sets the length field. The data is not null-terminated or blank-padded.
DT_FIXCHAR	Fixed-length blank-padded character string, in the CHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.
DT_LONGVARCHAR	Long varying length character string, in the CHAR character set.
DT_STRING	Null-terminated character string, in the CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.
DT_DOUBLE	8-byte floating-point number.
DT_FLOAT	4-byte floating-point number.
DT_DECIMAL	Packed decimal number (proprietary format).
DT_INT	32-bit signed integer.
DT_SMALLINT	16-bit signed integer.
DT_BINARY	Varying length binary data with a two-byte length field. The maximum length is 32765 bytes. When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.
DT_LONGBINARY	Long binary data.
DT_TINYINT	8-bit signed integer.
DT_BIGINT	64-bit signed integer.
DT_UNSYNINT	32-bit unsigned integer.
DT_UNSSMALLINT	16-bit unsigned integer.
DT_UNSBIGINT	64-bit unsigned integer.
DT_BIT	8-bit signed integer.
DT_LONGNVARCHAR	Long varying length character string, in the NCHAR character set.

**Remarks**

The value types correspond to the embedded SQL data types. For more information on embedded SQL data types, see [“Embedded SQL data types”](#) on page 449.

**See also**

- [“sqlany\\_get\\_column\\_info method \[SQL Anywhere C\]”](#) on page 541
- [“a\\_sqlany\\_column\\_info structure \[SQL Anywhere C\]”](#) on page 556

## a\_sqlany\_bind\_param structure

A bind parameter structure used to bind parameter and prepared statements.

**Syntax**

```
public typedef struct a_sqlany_bind_param
```

**Members**

Member name	Type	Description
direction	a_sqlany_data_direction	The direction of the data. (input, output, input_output)
name	char *	Name of the bind parameter. This is only used by sqlany_describe_bind_param().
value	a_sqlany_data_value	The actual value of the data.

**Remarks**

To view examples of the a\_sqlany\_bind\_param structure in use, see any of the following sample files in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

**See also**

- [“sqlany\\_execute method \[SQL Anywhere C\]”](#) on page 534

## a\_sqlany\_bind\_param\_info structure

Gets information about the currently bound parameters.

**Syntax**

```
public typedef struct a_sqlany_bind_param_info
```

**Members**

Member name	Type	Description
direction	a_sqlany_data_direction	The direction of the parameter.
input_value	a_sqlany_data_value	Information about the bound input value.
name	char *	A pointer to the name of the parameter.
output_value	a_sqlany_data_value	Information about the bound output value.

**Remarks**

sqlany\_get\_bind\_param\_info() can be used to populate this structure.

To view examples of the a\_sqlany\_bind\_param\_info structure in use, see any of the following sample files in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

**See also**

- [“sqlany\\_execute method \[SQL Anywhere C\]” on page 534](#)

## a\_sqlany\_column\_info structure

Returns column metadata information.

**Syntax**

```
public typedef struct a_sqlany_column_info
```

**Members**

Member name	Type	Description
max_size	size_t	The maximum size a data value in this column can take.
name	char *	The name of the column (null-terminated). The string can be referenced as long as the result set object is not freed.
native_type	a_sqlany_native_type	The native type of the column in the database.
nullable	sacapi_bool	Indicates whether a value in the column can be null.
precision	unsigned short	The precision.
scale	unsigned short	The scale.

Member name	Type	Description
type	a_sqlany_data_type	The column data type.

**Remarks**

sqlany\_get\_column\_info() can be used to populate this structure.

To view an example of the a\_sqlany\_column\_info structure in use, see the following sample file in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

- [dbcapi\\_isql.cpp](#)

## a\_sqlany\_data\_info structure

Returns metadata information about a column value in a result set.

**Syntax**

```
public typedef struct a_sqlany_data_info
```

**Members**

Member name	Type	Description
data_size	size_t	The total number of bytes available to be fetched. This field is only valid after a successful fetch operation.
is_null	sacapi_bool	Indicates whether the last fetched data is NULL. This field is only valid after a successful fetch operation.
type	a_sqlany_data_type	The type of the data in the column.

**Remarks**

sqlany\_get\_data\_info() can be used to populate this structure with information about what was last retrieved by a fetch operation.

To view an example of the a\_sqlany\_data\_data\_info structure in use, see the following sample file in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

**See also**

- [“sqlany\\_get\\_data\\_info method \[SQL Anywhere C\]” on page 542](#)

## a\_sqlany\_data\_value structure

Returns a description of the attributes of a data value.

### Syntax

```
public typedef struct a_sqlany_data_value
```

### Members

Member name	Type	Description
buffer	char *	A pointer to user supplied buffer of data.
buffer_size	size_t	The size of the buffer.
is_null	sacapi_bool *	A pointer to indicate whether the last fetched data is NULL.
length	size_t *	A pointer to the number of valid bytes in the buffer. This value must be less than buffer_size.
type	a_sqlany_data_type	The type of the data.

### Remarks

To view examples of the a\_sqlany\_data\_value structure in use, see any of the following sample files in the *sdk\dbcapi\examples* directory of your SQL Anywhere installation.

- *dbcapi\_isql.cpp*
- *fetching\_a\_result\_set.cpp*
- *send\_retrieve\_full\_blob.cpp*
- *preparing\_statements.cpp*

## SQLAnywhereInterface structure

The SQL Anywhere C API interface structure.

### Syntax

```
public typedef struct SQLAnywhereInterface
```



**Members**

<b>Member name</b>	<b>Type</b>	<b>Description</b>
dll_handle	void *	DLL handle.
function	method	Pointer to sqlany_init() function.
function	method	Pointer to sqlany_fini() function.
function	method	Pointer to sqlany_new_connection() function.
function	method	Pointer to sqlany_free_connection() function.
function	method	Pointer to sqlany_make_connection() function.
function	method	Pointer to sqlany_connect() function.
function	method	Pointer to sqlany_disconnect() function.
function	method	Pointer to sqlany_execute_immediate() function.
function	method	Pointer to sqlany_prepare() function.
function	method	Pointer to sqlany_free_stmt() function.
function	method	Pointer to sqlany_num_params() function.
function	method	Pointer to sqlany_describe_bind_param() function.
function	method	Pointer to sqlany_bind_param() function.
function	method	Pointer to sqlany_send_param_data() function.
function	method	Pointer to sqlany_reset() function.
function	method	Pointer to sqlany_get_bind_param_info() function.
function	method	Pointer to sqlany_execute() function.
function	method	Pointer to sqlany_execute_direct() function.
function	method	Pointer to sqlany_fetch_absolute() function.
function	method	Pointer to sqlany_fetch_next() function.
function	method	Pointer to sqlany_get_next_result() function.
function	method	Pointer to sqlany_affected_rows() function.

Member name	Type	Description
function	method	Pointer to sqlany_num_cols() function.
function	method	Pointer to sqlany_num_rows() function.
function	method	Pointer to sqlany_get_column() function.
function	method	Pointer to sqlany_get_data() function.
function	method	Pointer to sqlany_get_data_info() function.
function	method	Pointer to sqlany_get_column_info() function.
function	method	Pointer to sqlany_commit() function.
function	method	Pointer to sqlany_rollback() function.
function	method	Pointer to sqlany_client_version() function.
function	method	Pointer to sqlany_error() function.
function	method	Pointer to sqlany_sqlstate() function.
function	method	Pointer to sqlany_clear_error() function.
function	method	Pointer to sqlany_init_ex() function.
function	method	Pointer to sqlany_fini_ex() function.
function	method	Pointer to sqlany_new_connection_ex() function.
function	method	Pointer to sqlany_make_connection_ex() function.
function	method	Pointer to sqlany_client_version_ex() function.
function	method	Pointer to sqlany_cancel() function.
initialized	int	Flag to know if initialized or not.

**See also**

- [“sqlany\\_initialize\\_interface method \[SQL Anywhere C\]” on page 545](#)

**SACAPI\_ERROR\_SIZE variable**

Returns the minimal error buffer size.

**Syntax**

```
#define SACAPI_ERROR_SIZE
```

**SQLANY\_API\_VERSION\_1 variable**

Defines to indicate the API versions.

**Syntax**

```
#define SQLANY_API_VERSION_1
```

**SQLANY\_API\_VERSION\_2 variable**

Version 2 introduced the "\_ex" functions and the ability to cancel requests.

**Syntax**

```
#define SQLANY_API_VERSION_2
```

---

---

# SQL Anywhere external call interface

You can call a function in an external library from a stored procedure or function. You can call functions in a DLL under Windows operating systems and in a shared object on Unix. You cannot call external functions on Windows Mobile.

This section describes how to use the external function call interface. Sample external stored procedures, plus the files required to build a DLL containing them, are located in the following folder:  
`%SQLANYSAMPI2%\SQLAnywhere\ExternalProcedures.`

## Caution

External libraries called from procedures share the memory of the server. If you call an external library from a procedure and the external library contains memory-handling errors, you can crash the server or corrupt your database. Ensure that you thoroughly test your libraries before deploying them on production databases.

The interface described in this section replaces an older interface, which has been deprecated. Libraries written to the older interface, used in versions before version 7.0.x, are still supported, but in any new development, the new interface is recommended. Note that the new interface must be used for all Unix platforms and for all 64-bit platforms, including 64-bit Windows.

SQL Anywhere includes a set of system procedures that make use of this capability, for example to send MAPI email messages. See “[MAPI and SMTP procedures](#)” [[SQL Anywhere Server - SQL Reference](#)].

## Procedures and functions that use external calls

This section presents some examples of procedures and functions with external calls.

## Note

You must have DBA authority to create procedures or functions that reference external libraries. This requirement is more strict than the RESOURCE authority required for creating other procedures or functions.

## Syntax

You can create a SQL stored procedure that calls a C/C++ function in a library (a Dynamic Link Library (DLL) or shared object) as follows:

```
CREATE PROCEDURE coverProc( parameter-list )
  EXTERNAL NAME 'myFunction@myLibrary'
  LANGUAGE C_ESQL32;
```

When you define a stored procedure or function in this way, you are creating a bridge to the function in the external DLL. The stored procedure or function cannot perform any other tasks.

Similarly, you can create a SQL stored function that calls a C/C++ function in a library as follows:

```
CREATE FUNCTION coverFunc( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'myFunction@myLibrary'
  LANGUAGE C_ESQL32;
```

In these statements, the `EXTERNAL NAME` clause indicates the function name and library in which it resides. In the example, `myFunction` is the exported name of a function in the library, and `myLibrary` is the name of the library (for example, `myLibrary.dll` or `myLibrary.so`).

The `LANGUAGE` clause indicates that the function is to be called in an external environment. The `LANGUAGE` clause can specify one of `C_ESQL32`, `C_ESQL64`, `C_ODBC32`, or `C_ODBC64`. The **32** or **64** suffix indicates that the function is compiled as a 32-bit or 64-bit application. The **ODBC** designation indicates that the application uses the ODBC API. The **ESQL** designation indicates that the application could use the Embedded SQL API, the SQL Anywhere C API, any other non-ODBC API, or no API at all.

If the `LANGUAGE` clause is omitted, then the library containing the function is loaded into the address space of the database server. When called, the external function will execute as part of the server. In this case, if the function causes a fault, then the database server will be terminated. Because of this, loading and executing functions in an external environment is recommended. If a function causes a fault in an external environment, the database server will continue to run.

The arguments in *parameter-list* must correspond in type and order to the arguments expected by the library function. The library function accesses the procedure arguments using a special interface. See [“External function prototypes” on page 565](#).

Any value or result set returned by the external function can be returned by the stored procedure or function to the calling environment.

### No other statements permitted

A stored procedure or function that references an external function can include no other statements: its sole purpose is to take arguments for a function, call the function, and return any value and returned arguments from the function to the calling environment. You can use `IN`, `INOUT`, or `OUT` parameters in the procedure call in the same way as for other procedures: the input values get passed to the external function, and any parameters modified by the function are returned to the calling environment in `OUT` or `INOUT` parameters or as the `RETURNS` result of the stored function.

### System-dependent calls

You can specify operating-system dependent calls, so that a procedure calls one function when run on one operating system, and another function (presumably analogous) on another operating system. The syntax for such calls involves prefixing the function name with the operating system name. The operating system identifier must be `Unix`. An example follows.

```
CREATE FUNCTION func ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'Unix:function-name@library.so;function-name@library.dll';
```

If the list of functions does not contain an entry for the operating system on which the server is running, but the list does contain an entry without an operating system specified, the database server calls the function in that entry.

**See also**

- “CREATE FUNCTION statement [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “CREATE FUNCTION statement [External call]” [*SQL Anywhere Server - SQL Reference*]
- “SQL Anywhere external environment support” on page 581

## External function prototypes

This section describes the interface that you use for functions written in C or C++.

The interface is defined by a header file named *extfnapi.h*, in the *SDK\Include* subdirectory of your SQL Anywhere installation directory. This header file handles the platform-dependent features of external function prototypes.

**Function prototypes**

The name of the function must match that referenced in the CREATE PROCEDURE or CREATE FUNCTION statement. Suppose the following CREATE FUNCTION statement had been executed.

```
CREATE FUNCTION cover-name ( parameter-list )
  RETURNS data-type
  EXTERNAL NAME 'function-name@library.dll'
  LANGUAGE C_ESQL32;
```

The C/C++ function declaration must be as follows:

```
void function-name( an_extfn_api *api, void *argument-handle )
```

The function must return void, and must take as arguments a pointer to a structure used to call a set of callback functions and a handle to the arguments provided by the SQL procedure.

## extfn\_use\_new\_api method

To notify the database server that the external library is written using the external function call interface, your external library must export the following function:

**Syntax**

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void );
```

**Returns**

The function returns an unsigned 32-bit integer. The returned value must be the interface version number, `EXTFN_API_VERSION`, defined in *extfnapi.h*. A return value of 0 means that the old, deprecated interface is being used.

## Remarks

If the function is not exported by the library, the database server assumes that the old interface is in use. The new interface must be used for all Unix platforms and for all 64-bit platforms, including 64-bit Windows.

A typical implementation of this function follows:

```
extern "C" a_sql_uint32 SQL_CALLBACK extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}
```

## See also

- [“an\\_extfn\\_api structure” on page 568](#)

## extfn\_cancel method

To notify the database server that the external library supports cancel processing, your external library must export the following function:

### Syntax

```
extern "C" void SQL_CALLBACK extfn_cancel( void *cancel_handle );
```

### Parameters

- **cancel\_handle** A pointer to a variable to manipulate.

## Remarks

This function is called asynchronously by the database server whenever the currently executing SQL statement is canceled.

The function uses the cancel\_handle to set a flag indicating to the external library functions that the SQL statement has been canceled.

If the function is not exported by the library, the database server assumes that cancel processing is not supported.

A typical implementation of this function follows:

```
extern "C" void SQL_CALLBACK extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}
```

## See also

- [“an\\_extfn\\_api structure” on page 568](#)



## extfn\_post\_load\_library method

If this function is implemented and exposed in the external library, it is executed by the database server after the external library has been loaded and the version check has been performed, and before any other function defined in the external library is called.

### Syntax

```
extern "C" void SQL_CALLBACK extfn_post_load_library( void );
```

### Remarks

This function is required only if there is a library-specific requirement to do library-wide setup before any function within the library is called.

This function is called asynchronously by the database server after the external library has been loaded and the version check has been performed, and before any other function defined in the external library is called.

### Example

```
extern "C" __declspec( dllexport )
void SQL_CALLBACK extfn_post_load_library( void )
{
    MessageBox( NULL, "Library loaded",
               "SQL Anywhere",
               MB_OK | MB_TASKMODAL );
}
```

### See also

- [“an\\_extfn\\_api structure” on page 568](#)

## extfn\_pre\_unload\_library method

If this function is implemented and exposed in the external library, it is executed by the database server immediately before unloading the external library.

### Syntax

```
extern "C" void SQL_CALLBACK extfn_pre_unload_library( void );
```

### Remarks

This function is required only if there is a library-specific requirement to do library-wide cleanup before the library is unloaded.

This function is called asynchronously by the database server immediately before unloading the external library.

### Example

```
extern "C" __declspec( dllexport )
void SQL_CALLBACK extfn_pre_unload_library( void )
```

```
{
    MessageBox( NULL, "Library unloading",
               "SQL Anywhere",
               MB_OK | MB_TASKMODAL );
}
```

**See also**

- “[an\\_extfn\\_api structure](#)” on page 568

## an\_extfn\_api structure

Used to communicate with the calling SQL environment.

**Syntax**

```
typedef struct an_extfn_api {
    short (SQL_CALLBACK *get_value)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value
    );
    short (SQL_CALLBACK *get_piece)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value,
        a_sql_uint32 offset
    );
    short (SQL_CALLBACK *set_value)(
        void *      arg_handle,
        a_sql_uint32 arg_num,
        an_extfn_value *value
        short      append
    );
    void (SQL_CALLBACK *set_cancel)(
        void *      arg_handle,
        void *      cancel_handle
    );
} an_extfn_api;
```

**Properties**

- **get\_value** Use this callback function to get the specified parameter's value. The following example gets the value for parameter 1.

```
result = extapi->get_value( arg_handle, 1, &arg )
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}
```

- **get\_piece** Use this callback function to get the next chunk of the specified parameter's value (if there are any). The following example gets the remaining pieces for parameter 1.

```
cmd = (char *)malloc( arg.len.total_len + 1 );
offset = 0;
for( ; result != 0; )
{
    if( arg.data == NULL ) break;
```

```

        memcpy( &cmd[offset], arg.data, arg.piece_len );
        offset += arg.piece_len;
        cmd[offset] = '\\0';
        if( arg.piece_len == 0 ) break;
        result = extapi->get_piece( arg_handle, 1, &arg, offset );
    }

```

- **set\_value** Use this callback function to set the specified parameter's value. The following example sets the return value (parameter 0) for a RETURNS clause of a FUNCTION.

```

    an_extfn_value      retval;
    int ret = -1;

    // set up the return value struct
    retval.type = DT_INT;
    retval.data = (void*) &ret;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );
    extapi->set_value( arg_handle, 0, &retval, 0 );

```

- **set\_cancel** Use this callback function to establish a pointer to a variable that can be set by the **extfn\_cancel** method. The following is example.

```

    short               canceled = 0;
    extapi->set_cancel( arg_handle, &canceled );

```

## Remarks

A pointer to the `an_extfn_api` structure is passed by the caller to your external function. Here is an example.

```

extern "C" __declspec( dllexport )
void my_external_proc( an_extfn_api *extapi, void *arg_handle )
{
    short               result;
    short               canceled;
    an_extfn_value      arg;

    canceled = 0;
    extapi->set_cancel( arg_handle, &canceled );

    result = extapi->get_value( arg_handle, 1, &arg );
    if( canceled || result == 0 || arg.data == NULL )
    {
        return; // no parameter or parameter is NULL
    }
    .
    .
    .
}

```

Whenever you use any of the callback functions, you must pass back the argument handle that was passed to your external function as the second parameter.

## See also

- [“an\\_extfn\\_value structure” on page 570](#)
- [“extfn\\_cancel method” on page 566](#)

## an\_extfn\_value structure

Used to access parameter data from the calling SQL environment.

### Syntax

```
typedef struct an_extfn_value {
    void *      data;
    a_sql_uint32  piece_len;
    union {
        a_sql_uint32 total_len;
        a_sql_uint32 remain_len;
    } len;
    a_sql_data_type  type;
} an_extfn_value;
```

### Properties

- **data** A pointer to the data for this parameter.
- **piece\_len** The length of this segment of the parameter. This is less than or equal to **total\_len**.
- **total\_len** The total length of the parameter. For strings, this represents the length of the string and does not include a null terminator. This property is set after a call to the **get\_value** callback function. This property is no longer valid after a call to the **get\_piece** callback function.
- **remain\_len** When the parameter is obtained in segments, this is the length of the part that has not yet been obtained. This property is set after each call to the **get\_piece** callback function.
- **type** Indicates the type of the parameter. This is one of the Embedded SQL data types such as **DT\_INT**, **DT\_FIXCHAR**, or **DT\_BINARY**. See [“Embedded SQL data types” on page 449](#).

### Remarks

Suppose that your external function interface was described using the following SQL statement.

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@c:\\project\\mystring.dll';
```

The following code fragment shows how to access the properties for objects of type **an\_extfn\_value**. In the example, the input parameter 1 (**instr**) to this function (**mystring**) is expected to be a SQL **LONGVARCHAR** string.

```
an_extfn_value      arg;

result = extapi->get_value( arg_handle, 1, &arg );
if( result == 0 || arg.data == NULL )
{
    return; // no parameter or parameter is NULL
}

if( arg.type != DT_LONGVARCHAR )
{
    return; // unexpected type of parameter
}
```

```

cmd = (char *)malloc( arg.len.total_len + 1 );
offset = 0;
for( ; result != 0; )
{
    if( arg.data == NULL ) break;
    memcpy( &cmd[offset], arg.data, arg.piece_len );
    offset += arg.piece_len;
    cmd[offset] = '\0';
    if( arg.piece_len == 0 ) break;
    result = extapi->get_piece( arg_handle, 1, &arg, offset );
}

```

**See also**

- [“an\\_extfn\\_api structure” on page 568](#)

## an\_extfn\_result\_set\_info structure

Facilitates the return of result sets to the calling SQL environment.

**Syntax**

```

typedef struct an_extfn_result_set_info {
    a_sql_uint32                number_of_columns;
    an_extfn_result_set_column_info *column_infos;
    an_extfn_result_set_column_data *column_data_values;
} an_extfn_result_set_info;

```

**Properties**

- **number\_of\_columns** The number of columns in the result set.
- **column\_infos** Link to a description of the result set columns. See [“an\\_extfn\\_result\\_set\\_column\\_info structure” on page 572](#).
- **column\_data\_values** Link to a description of the result set column data. See [“an\\_extfn\\_result\\_set\\_column\\_data structure” on page 573](#).

**Remarks**

The following code fragment shows how to set the properties for objects of this type.

```

int columns = 2;
an_extfn_result_set_info    rs_info;

an_extfn_result_set_column_info *col_info =
    (an_extfn_result_set_column_info *)
    malloc( columns * sizeof(an_extfn_result_set_column_info) );

an_extfn_result_set_column_data *col_data =
    (an_extfn_result_set_column_data *)
    malloc( columns * sizeof(an_extfn_result_set_column_data) );

rs_info.number_of_columns    = columns;
rs_info.column_infos        = col_info;
rs_info.column_data_values  = col_data;

```

**See also**

- [“an\\_extfn\\_result\\_set\\_column\\_info structure” on page 572](#)
- [“an\\_extfn\\_result\\_set\\_column\\_data structure” on page 573](#)

## an\_extfn\_result\_set\_column\_info structure

Used to describe a result set.

**Syntax**

```
typedef struct an_extfn_result_set_column_info {
    char *                column_name;
    a_sql_data_type       column_type;
    a_sql_uint32          column_width;
    a_sql_uint32          column_index;
    short int             column_can_be_null;
} an_extfn_result_set_column_info;
```

**Properties**

- **column\_name** Points to the name of the column which is a null-terminated string.
- **column\_type** Indicates the type of the column. This is one of the Embedded SQL data types such as **DT\_INT**, **DT\_FIXCHAR**, or **DT\_BINARY**. See [“Embedded SQL data types” on page 449](#).
- **column\_width** Defines the maximum width for char(n), varchar(n) and binary(n) declarations and is set to 0 for all other types.
- **column\_index** The ordinal position of the column which starts at 1.
- **column\_can\_be\_null** Set to 1 if the column is nullable; otherwise it is set to 0.

**Remarks**

The following code fragment shows how to set the properties for objects of this type and how to describe the result set to the calling SQL environment.

```
// set up column descriptions
// DepartmentID          INTEGER NOT NULL
col_info[0].column_name = "DepartmentID";
col_info[0].column_type = DT_INT;
col_info[0].column_width = 0;
col_info[0].column_index = 1;
col_info[0].column_can_be_null = 0;

// DepartmentName       CHAR(40) NOT NULL
col_info[1].column_name = "DepartmentName";
col_info[1].column_type = DT_FIXCHAR;
col_info[1].column_width = 40;
col_info[1].column_index = 2;
col_info[1].column_can_be_null = 0;

extapi->set_value( arg_handle,
                  EXTFN_RESULT_SET_ARG_NUM,
                  (an_extfn_value *)&rs_info,
                  EXTFN_RESULT_SET_DESCRIBE );
```

**See also**

- [“an\\_extfn\\_result\\_set\\_info structure” on page 571](#)
- [“an\\_extfn\\_result\\_set\\_column\\_data structure” on page 573](#)

## an\_extfn\_result\_set\_column\_data structure

Used to return the data values for columns.

**Syntax**

```
typedef struct an_extfn_result_set_column_data {
    a_sql_uint32      column_index;
    void *           column_data;
    a_sql_uint32      data_length;
    short            append;
} an_extfn_result_set_column_data;
```

**Properties**

- **column\_index** The ordinal position of the column which starts at 1.
- **column\_data** Pointer to a buffer containing the column data.
- **data\_length** The actual length of the data.
- **append** Used to return the column value in chunks. Set to 1 when returning a partial column value; 0 otherwise.

**Remarks**

The following code fragment shows how to set the properties for objects of this type and how to return the result set row to the calling SQL environment.

```
int DeptNumber = 400;
char * DeptName = "Marketing";

col_data[0].column_index = 1;
col_data[0].column_data = &DeptNumber;
col_data[0].data_length = sizeof( DeptNumber );
col_data[0].append = 0;

col_data[1].column_index = 2;
col_data[1].column_data = DeptName;
col_data[1].data_length = strlen(DeptName);
col_data[1].append = 0;

extapi->set_value( arg_handle,
                  EXTFN_RESULT_SET_ARG_NUM,
                  (an_extfn_value *)&rs_info,
                  EXTFN_RESULT_SET_NEW_ROW_FLUSH );
```

**See also**

- [“an\\_extfn\\_result\\_set\\_info structure” on page 571](#)
- [“an\\_extfn\\_result\\_set\\_column\\_info structure” on page 572](#)

## External function call interface methods

### get\_value callback

```
short (SQL_CALLBACK *get_value)
(
    void *      arg_handle,
    a_sql_uint32 arg_num,
    an_extfn_value *value
);
```

The **get\_value** callback function can be used to obtain the value of a parameter that was passed to the stored procedure or function that acts as the interface to the external function. It returns 0 if not successful; otherwise it returns a non-zero result. After calling `get_value`, the `total_len` field of the `an_extfn_value` structure contains the length of the entire value. The `piece_len` field contains the length of the portion that was obtained as a result of calling `get_value`. Note that `piece_len` will always be less than or equal to `total_len`. When it is less than, a second function **get\_piece** can be called to obtain the remaining pieces. Note that the `total_len` field is only valid after the initial call to `get_value`. This field is overlaid by the `remain_len` field which is altered by calls to `get_piece`. It is important to preserve the value of the `total_len` field immediately after calling `get_value` if you plan to use it later on.

### get\_piece callback

```
short (SQL_CALLBACK *get_piece)
(
    void *      arg_handle,
    a_sql_uint32 arg_num,
    an_extfn_value *value,
    a_sql_uint32 offset
);
```

If the entire parameter value cannot be returned in one piece, then the **get\_piece** function can be called iteratively to obtain the remaining pieces of the parameter value.

The sum of all the `piece_len` values returned by both calls to `get_value` and `get_piece` will add up to the initial value that was returned in the `total_len` field after calling `get_value`. After calling `get_piece`, the `remain_len` field, which overlays `total_len`, represents the amount not yet obtained.

### Using get\_value and get\_piece callbacks

The following example shows the use of `get_value` and `get_piece` to obtain the value of a string parameter such as a long varchar parameter.

Suppose that the wrapper to an external function was declared as follows:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mystring.dll';
```

To call the external function from SQL, use a statement like the following.

```
call mystring('Hello world!');
```

A sample implementation for the Windows operating system of the `mystring` function, written in C, follows:



```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD   ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    return TRUE;
}

extern "C" __declspec( dllexport )
a_sql_uint32 extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}

extern "C" __declspec( dllexport )
void mystring( an_extfn_api *extapi, void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    unsigned       offset;
    char           *string;

    result = extapi->get_value( arg_handle, 1, &arg );
    if( result == 0 || arg.data == NULL )
    {
        return; // no parameter or parameter is NULL
    }
    string = (char *)malloc( arg.len.total_len + 1 );
    offset = 0;
    for( ; result != 0; ) {
        if( arg.data == NULL ) break;
        memcpy( &string[offset], arg.data, arg.piece_len );
        offset += arg.piece_len;
        string[offset] = '\\0';
        if( arg.piece_len == 0 ) break;
        result = extapi->get_piece( arg_handle, 1, &arg, offset );
    }
    MessageBoxA( NULL, string,
                "SQL Anywhere",
                MB_OK | MB_TASKMODAL );
    free( string );
    return;
}

```

### set\_value callback

```

short (SQL_CALLBACK *set_value)
(
    void *          arg_handle,
    a_sql_uint32   arg_num,
    an_extfn_value *value
    short          append
);

```

The `set_value` callback function can be used to set the values of OUT parameters and the RETURNS result of a stored function. Use an `arg_num` value of 0 to set the RETURNS value. The following is an example.

```
an_extfn_value      retval;

retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.piece_len = retval.len.total_len = (a_sql_uint32) strlen( result );
extapi->set_value( arg_handle, 0, &retval, 0 );
```

The `append` argument of `set_value` determines whether the supplied data replaces (false) or appends to (true) the existing data. You must call `set_value` with `append=FALSE` before calling it with `append=TRUE` for the same argument. The `append` argument is ignored for fixed-length data types.

To return NULL, set the data field of the `an_extfn_value` structure to NULL.

### set\_cancel callback

```
void (SQL_CALLBACK *set_cancel)
(
    void *arg_handle,
    void *cancel_handle
);
```

External functions can get the values of IN or INOUT parameters and set the values of OUT parameters and the RETURNS result of a stored function. There is a case, however, where the parameter values obtained may no longer be valid or the setting of values is no longer necessary. This occurs when an executing SQL statement is canceled. This may occur as the result of an application abruptly disconnecting from the database server. To handle this situation, you can define a special entry point in the library called `extfn_cancel`. When this function is defined, the server will call it whenever a running SQL statement is canceled.

The `extfn_cancel` function is called with a handle that can be used in any way you consider suitable. A typical use of the handle is to indirectly set a flag to indicate that the calling SQL statement has been canceled.

The value of the handle that is passed can be set by functions in the external library using the `set_cancel` callback function. This is illustrated by the following code fragment.

```
extern "C" __declspec( dllexport )
void extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}

extern "C" __declspec( dllexport )
void mystring( an_extfn_api *api, void *arg_handle )
{
    .
    .
    .
    short canceled = 0;

    extapi->set_cancel( arg_handle, &canceled );
    .
}
```

```

.
.
    if( canceled )

```

Note that setting a static global "canceled" variable is inappropriate since that would be misinterpreted as all SQL statements on all connections being canceled which is usually not the case. This is why a `set_cancel` callback function is provided. Make sure to initialize the "canceled" variable before calling `set_cancel`.

It is important to check the setting of the "canceled" variable at strategic points in your external function. Strategic points would include before and after calling any of the external library call interface functions like `get_value` and `set_value`. When the variable is set (as a result of `extfn_cancel` having been called), then the external function can take appropriate termination action. A code fragment based on the earlier example follows:

```

if( canceled )
{
    free( string );
    return;
}

```

## Notes

The `get_piece` function for any given argument can only be called immediately after the `get_value` function for the same argument.

Calling `get_value` on an OUT parameter returns the type field of the `an_extfn_value` structure set to the data type of the argument, and returns the data field of the `an_extfn_value` structure set to NULL.

The header file `extfnapi.h` in the SQL Anywhere installation `SDK\Include` folder contains some additional notes.

The following table shows the conditions under which the functions defined in `an_extfn_api` return false:

Function	Returns 0 when the following is true; else returns 1
<code>get_value()</code>	<ul style="list-style-type: none"> <li>• <code>arg_num</code> is invalid; for example, <code>arg_num</code> is greater than the number of arguments for the external function.</li> </ul>
<code>get_piece()</code>	<ul style="list-style-type: none"> <li>• <code>arg_num</code> is invalid; for example, <code>arg_num</code> does not correspond to the argument number used with the previous call to <code>get_value</code>.</li> <li>• The offset is greater than the total length of the value for the <code>arg_num</code> argument.</li> <li>• It is called before <code>get_value</code> has been called.</li> </ul>

Function	Returns 0 when the following is true; else returns 1
set_value()	<ul style="list-style-type: none"> <li>• arg_num is invalid; for example, arg_num is greater than the number of arguments for the external function.</li> <li>• Argument arg_num is input only.</li> <li>• The type of value supplied does not match that of argument arg_num.</li> </ul>

## Data type handling

### Data types

The following SQL data types can be passed to an external library:

SQL data type	sqldef.h	C type
CHAR	DT_FIXCHAR	Character data, with a specified length
VARCHAR	DT_VARCHAR	Character data, with a specified length
LONG VARCHAR, TEXT	DT_LONG-VARCHAR	Character data, with a specified length
UNIQUEIDENTIFIERSTR	DT_FIXCHAR	Character data, with a specified length
XML	DT_LONG-VARCHAR	Character data, with a specified length
NCHAR	DT_NFIX-CHAR	UTF-8 character data, with a specified length
NVARCHAR	DT_NVARCH-AR	UTF-8 character data, with a specified length
LONG NVARCHAR, NTEXT	DT_LONG-NVARCHAR	UTF-8 character data, with a specified length
UNIQUEIDENTIFIER	DT_BINARY	Binary data, 16 bytes long
BINARY	DT_BINARY	Binary data, with a specified length
VARBINARY	DT_BINARY	Binary data, with a specified length
LONG BINARY	DT_LONGBI-NARY	Binary data, with a specified length

SQL data type	sqldef.h	C type
TINYINT	DT_TINYINT	1-byte integer
[ UNSIGNED ] SMALLINT	DT_SMALLINT, DT_UNSSMALLINT	[ Unsigned ] 2-byte integer
[ UNSIGNED ] INT	DT_INT, DT_UNSENT	[ Unsigned ] 4-byte integer
[ UNSIGNED ] BIGINT	DT_BIGINT, DT_UNSBIGINT	[ Unsigned ] 8-byte integer
REAL, FLOAT(1-24)	DT_FLOAT	Single-precision floating-point number
DOUBLE, FLOAT(25-53)	DT_DOUBLE	Double-precision floating-point number

You cannot use any of the date or time data types, and you cannot use the DECIMAL or NUMERIC data types (including the money types).

To provide values for INOUT or OUT parameters, use the `set_value` function. To read IN and INOUT parameters, use the `get_value` function.

### Determining data types of parameters

After a call to `get_value`, the type field of the `an_extfn_value` structure can be used to obtain data type information for the parameter. The following sample code fragment shows how to identify the type of the parameter.

```

an_extfn_value    arg;
a_sql_data_type  data_type;

extapi->get_value( arg_handle, 1, &arg );
data_type = arg.type & DT_TYPES;
switch( data_type )
{
case DT_FIXCHAR:
case DT_VARCHAR:
case DT_LONGVARCHAR:
    break;
default:
    return;
}

```

For more information about data types, see [“Host variables in embedded SQL”](#) on page 452.

### UTF-8 types

The UTF-8 data types such as NCHAR, NVARCHAR, LONG NVARCHAR and NTEXT are passed as UTF-8 encoded strings. A function such as the Windows `MultiByteToWideChar` function can be used to convert a UTF-8 string to a wide-character (Unicode) string.

## Passing NULL

You can pass NULL as a valid value for all arguments. Functions in external libraries can supply NULL as a return value for any data type.

## Return values

To set a return value in an external function, call the `set_value` function with an `arg_num` parameter value of 0. If `set_value` is not called with `arg_num` set to 0, the function result is NULL.

It is also important to set the data type of a return value for a stored function call. The following code fragment shows how to set the return data type.

```
an_extfn_value      retval;

retval.type = DT_LONGVARCHAR;
retval.data = result;
retval.piece_len = retval.len.total_len = (a_sql_uint32) strlen( result );
extapi->set_value( arg_handle, 0, &retval, 0 );
```

# Unloading external libraries

The system procedure, `dbo.sa_external_library_unload`, can be used to unload an external library when the library is not in use. The procedure takes one optional parameter, a long varchar. The parameter specifies the name of the library to be unloaded. If no parameter is specified, all external libraries not in use will be unloaded.

The following example unloads an external function library.

```
CALL sa_external_library_unload( 'library.dll' );
```

This function is useful when developing a set of external functions because you do not have to shut down the database server to install a newer version of the library.

---

# SQL Anywhere external environment support

SQL Anywhere includes support for six external runtime environments. These include embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

SQL Anywhere has had the ability to call compiled native functions written in C or C++ for some time. However, when these procedures are run by the database server, the dynamic link library or shared object has always been loaded by the database server and the calls out to the native functions have always been made by the database server. The risk here is that if the native function causes a fault, then the database server crashes. Running compiled native functions outside the database server, in an external environment, eliminates these risks to the server.

The following is an overview of the external environment support in SQL Anywhere.

## Catalog tables

A system catalog table stores the information needed to identify and launch each of the external environments. The definition for this table is:

```
SYS.SYSEXTERNENV (
  object_id      unsigned bigint not null,
  name           varchar(128)   not null,
  scope         char(1)        not null,
  supports_result_sets char(1)   not null,
  location      long varchar   not null,
  options       long varchar   not null,
  user_id       unsigned int
)
```

- **object\_id** A unique identifier that is generated by the database server.
- **name** The name column identifies the name of the external environment or language. It is one of **java**, **perl**, **php**, **clr**, **c\_esql32**, **c\_esql64**, **c\_odbc32**, or **c\_odbc64**.
- **scope** The scope column is either **C** for CONNECTION or **D** for DATABASE respectively. The scope column identifies if the external environment is launched as one-per-connection or one-per-database.

For one-per-connection external environments (like PERL, PHP, C\_ESQL32, C\_ESQL64, C\_ODBC32, and C\_ODBC64), there is one instance of the external environment for each connection using the external environment. For a one-per-connection, the external environment terminates when the connection terminates.

For one-per-database external environments (like JAVA and CLR), there is one instance of the external environment for each database using the external environment. The one-per-database external environment terminates when the database is stopped.

- **supports\_result\_sets** The supports\_result\_sets column identifies those external environments that can return result sets. All external environments can return result sets except PERL and PHP.

- **location** The location column identifies the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the database server can find it.
- **options** The options column identifies any options required on the command line to launch the executable associated with the external environment. You should not modify this column.
- **user\_id** The user\_id column identifies a user ID in the database that has DBA authority. When the external environment is initially launched, it must make a connection back to the database to set things up for the external environment's usage. By default, this connection is made using the DBA user ID, but if the database administrator prefers to have the external environment use a different user ID with DBA authority, then the user\_id column in the SYS.SYSEXTERNENV table would indicate that different user ID instead. Typically, this column in SYS.SYSEXTERNENV is NULL and the database server, by default, uses the DBA user ID.

Another system catalog table stores the non-Java external objects. The table definition for this table is:

```
SYS.SYSEXTERNENVOBJECT (  
  object_id      unsigned bigint not null,  
  extenv_id     unsigned bigint not null,  
  owner         unsigned int   not null,  
  name          long varchar  not null,  
  contents      long binary   not null,  
  update_time   timestamp    not null  
)
```

- **object\_id** A unique identifier that is generated by the database server.
- **extenv\_id** The extenv\_id identifies the external environment type (as stored in SYS.SYSEXTERNENV).
- **owner** The owner column identifies the creator/owner of the external object.
- **name** The name column is the name of the external object as specified in the INSTALL EXTERNAL OBJECT statement.
- **contents** The contents column contains the contents of the external object.
- **update\_time** The update\_time column represents the last time the object was modified (or installed).

### Deprecated options

With the introduction of the SYS.SYSEXTERNENV table, some Java-specific options have now been deprecated. These deprecated options are:

```
java_location  
java_main_userid
```

Applications that have been using these options to identify which specific Java VM to use or which user ID to use for installing classes and other Java-related administrative tasks should use the ALTER



---

EXTERNAL ENVIRONMENT statement instead to set the location and user\_id values in the SYS.SYSEXTERNENV table for Java.

### Locating external environments

The following SQL syntax allows you to set or modify the location of external environments in SYS.SYSEXTERNENV table.

```
ALTER EXTERNAL ENVIRONMENT environment-name
  [ LOCATION location-string ]
```

To add a comment for an external environment, you can execute:

```
COMMENT ON EXTERNAL ENVIRONMENT environment-name
  IS comment-string
```

### Installing and using external objects

Once an external environment is set up to be used on the database server, you can then install objects into the database and create stored procedures and functions that make use of these objects within the external environment. Installation, creation, and usage of these objects, stored procedures, and stored functions is very similar to the current method of installing Java classes and creating and using Java stored procedures and functions.

To install a Perl or PHP external object (for example, a Perl script) from a file or an expression into the database, you would need to execute an INSTALL EXTERNAL OBJECT statement similar to the following:

```
INSTALL EXTERNAL OBJECT object-name-string
  [ update-mode ]
  FROM { FILE file-path | VALUE expression }
  ENVIRONMENT environment-name
```

To add a comment for an installed Perl or PHP external object, you can execute:

```
COMMENT ON EXTERNAL [ENVIRONMENT] OBJECT object-name-string
  IS comment-string
```

To remove an installed Perl or PHP external object from the database, you would need to use a REMOVE EXTERNAL OBJECT statement:

```
REMOVE EXTERNAL OBJECT object-name-string
```

Once the external objects are installed in the database, they can be used within external stored procedure and function definitions (similar to the current mechanism for creating Java stored procedures and functions).

```
CREATE PROCEDURE procedure-name(...)
  EXTERNAL NAME '...'
  LANGUAGE environment-name

CREATE FUNCTION function-name(...)
  RETURNS ...
  EXTERNAL NAME '...'
  LANGUAGE environment-name
```

Once these stored procedures and functions are created, they can be used like any other stored procedure or function in the database. The database server, when encountering an external environment stored procedure or function, automatically launches the external environment (if it has not already been started), and sends over whatever information is needed to get the external environment to fetch the external object from the database and execute it. Any result sets or return values resulting from the execution are returned as needed.

If you want to start or stop an external environment on demand, you can use the `START EXTERNAL ENVIRONMENT` and `STOP EXTERNAL ENVIRONMENT` statements (similar to the current `START JAVA` and `STOP JAVA` statements):

```
START EXTERNAL ENVIRONMENT environment-name
STOP EXTERNAL ENVIRONMENT environment-name
```

### See also

- “`ALTER EXTERNAL ENVIRONMENT` statement” [[SQL Anywhere Server - SQL Reference](#)]
- “`INSTALL EXTERNAL OBJECT` statement” [[SQL Anywhere Server - SQL Reference](#)]
- “`REMOVE EXTERNAL OBJECT` statement” [[SQL Anywhere Server - SQL Reference](#)]
- “`COMMENT` statement” [[SQL Anywhere Server - SQL Reference](#)]
- “`CREATE FUNCTION` statement [External call]” [[SQL Anywhere Server - SQL Reference](#)]
- “`CREATE PROCEDURE` statement [External call]” [[SQL Anywhere Server - SQL Reference](#)]
- “`START EXTERNAL ENVIRONMENT` statement” [[SQL Anywhere Server - SQL Reference](#)]
- “`STOP EXTERNAL ENVIRONMENT` statement” [[SQL Anywhere Server - SQL Reference](#)]

## The CLR external environment

SQL Anywhere includes support for CLR stored procedures and functions. A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable). There is only one instance of this .NET executable per database. All connections executing CLR functions and stored procedures use the same .NET executable instance, but the namespaces for each connection are separate. Statics persist for the duration of the connection, but are not shareable across connections. Only .NET version 2.0 is supported.

To call an external CLR function or procedure, you define a corresponding stored procedure or function with an `EXTERNAL NAME` string defining which DLL to load and which function within the assembly to call. You must also specify `LANGUAGE CLR` when defining the stored procedure or function. An example declaration follows:

```
CREATE PROCEDURE clr_stored_proc(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    OUT p3 LONG VARCHAR)
EXTERNAL NAME 'MyCLRTest.dll::MyCLRTest.Run( int, ushort, out string )'
LANGUAGE CLR;
```

In this example, the stored procedure called `clr_stored_proc`, when executed, loads the DLL `MyCLRTest.dll` and calls the function `MyCLRTest.Run`. The `clr_stored_proc` procedure takes three SQL

parameters, two IN parameters, one of type INT and one of type UNSIGNED SMALLINT, and one OUT parameter of type LONG VARCHAR. On the .NET side, these three parameters translate to input arguments of type int and ushort and an output argument of type string. In addition to out arguments, the CLR function can also have ref arguments. A user must declare a ref CLR argument if the corresponding stored procedure has an INOUT parameter.

The following table lists the various CLR argument types and the corresponding suggested SQL data type:

CLR type	Recommended SQL data type
bool	bit
byte	tinyint
short	smallint
ushort	unsigned smallint
int	int
uint	unsigned int
long	bigint
ulong	unsigned bigint
decimal	numeric
float	real
double	double
DateTime	timestamp
string	long varchar
byte[]	long binary

The declaration of the DLL can be either a relative or absolute path. If the specified path is relative, then the external .NET executable searches the path, and other locations, for the DLL. The executable does not search the Global Assembly Cache (GAC) for the DLL.

Like the existing Java stored procedures and functions, CLR stored procedures and functions can make server-side requests back to the database, and they can return result sets. Also, like Java, any information output to Console.Out and Console.Error is automatically redirected to the database server messages window.

For more information about how to make server-side requests and how to return result sets from a CLR function or stored procedure, refer to the samples located in the `%SQLANY12%\SQLAnywhere\ExternalEnvironments\CLR` directory.

To use CLR in the database, make sure the database server is able to locate and start the CLR executable. You can verify if the database server is able to locate and start the CLR executable by executing the following statement:

```
START EXTERNAL ENVIRONMENT CLR;
```

If the database server fails to start CLR, then the database server is likely not able to locate the CLR executable. The CLR executable is `dbextclr12.exe`. Make sure that this file is present in the `%SQLANY12%\CE\Assembly\V2\CE\Assembly\V2\Bin32` or `%SQLANY12%\CE\Assembly\V2\Bin64` folder, depending on which version of the database server you are using.

Note that the `START EXTERNAL ENVIRONMENT CLR` statement is not necessary other than to verify that the database server can launch CLR executables. In general, making a CLR stored procedure or function call starts CLR automatically.

Similarly, the `STOP EXTERNAL ENVIRONMENT CLR` statement is not necessary to stop an instance of CLR since the instance automatically goes away when the connection terminates. However, if you are completely done with CLR and you want to free up some resources, then the `STOP EXTERNAL ENVIRONMENT CLR` statement releases the CLR instance for your connection.

Unlike the Perl, PHP, and Java external environments, the CLR environment does not require the installation of anything in the database. As a result, you do not need to execute any `INSTALL` statements before using the CLR external environment.

Here is an example of a function written in C# that can be run within an external environment.

```
public class StaticTest
{
    private static int val = 0;

    public static int GetValue() {
        val += 1;
        return val;
    }
}
```

When compiled into a dynamic link library, this function can be called from an external environment. An executable image called `dbextclr12.exe` is started by the database server and it loads the dynamic link library for you. Different versions of this executable are included with SQL Anywhere. For example, on Windows you may have both 32-bit and 64-bit executables. One is for use with the 32-bit version of the database server and the other for the 64-bit version of the database server.

To build this application into a dynamic link library using the Microsoft C# compiler, use a command like the following. The source code for the above example is assumed to reside in a file called `StaticTest.cs`.

```
csc /target:library /out:clrtest.dll StaticTest.cs
```

This command places the compiled code in a DLL called `clrtest.dll`. To call the compiled C# function, `GetValue`, a wrapper is defined as follows using Interactive SQL:

```
CREATE FUNCTION stc_get_value()  
RETURNS INT  
EXTERNAL NAME 'clrtest.dll::StaticTest.GetValue() int'  
LANGUAGE CLR;
```

For CLR, the EXTERNAL NAME string is specified in a single line of SQL. You may be required to include the path to the DLL as part of the EXTERNAL NAME string so that it can be located. For dependent assemblies (for example, if *myLib.dll* has code that calls functions in, or in some way depends on, *myOtherLib.dll*) then it is up to the .NET Framework to load the dependencies. The CLR External Environment will take care of loading the specified assembly, but extra steps might be required to ensure that dependent assemblies are loaded. One solution is to register all dependencies in the Global Assembly Cache (**GAC**) by using the Microsoft **gacutil** utility installed with the .NET Framework. For custom-developed libraries, **gacutil** requires that these be signed with a strong name key before they can be registered in the **GAC**.

To execute the sample compiled C# function, execute the following statement.

```
SELECT stc_get_value();
```

Each time the C# function is called, a new integer result is produced. The sequence of values returned is 1, 2, 3, and so on.

For more information and examples on using the CLR in the database support, refer to the examples located in the *%SQLANYSAMPI2%\SQLAnywhere\ExternalEnvironments\CLR* directory.

## The ESQL and ODBC external environments

SQL Anywhere has had the ability to call compiled native functions written in C or C++ for some time. However, when these procedures are run by the database server, the dynamic link library or shared object has always been loaded by the database server and the calls out to the native functions have always been made by the database server. While having the database server make these native calls is most efficient, there can be serious consequences if the native function misbehaves. In particular, if the native function enters an infinite loop, then the database server can hang, and if the native function causes a fault, then the database server crashes. As a result, you now have the option of running compiled native functions outside the database server, in an external environment. There are some key benefits to running a compiled native function in an external environment:

1. The database server does not hang or crash if the compiled native function misbehaves.
2. The native function can be written to use ODBC, Embedded SQL (ESQL), or the SQL Anywhere C API and can make server-side calls back into the database server without having to make a connection.
3. The native function can return a result set to the database server.
4. In the external environment, a 32-bit database server can communicate with a 64-bit compiled native function and vice versa. Note that this is not possible when the compiled native functions are loaded directly into the address space of the database server. A 32-bit library can only be loaded by a 32-bit server and a 64-bit library can only be loaded by a 64-bit server.

Running a compiled native function in an external environment instead of within the database server results in a small performance penalty.

Also, the compiled native function must use the native function call interface to pass information to and return information from the native function. This interface is described in [“SQL Anywhere external call interface” on page 563](#).

To run a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute specifying one of C\_ESQL32, C\_ESQL64, C\_ODBC32, or C\_ODBC64.

Unlike the Perl, PHP, and Java external environments, you do not install any source code or compiled objects in the database. As a result, you do not need to execute any INSTALL statements before using the ESQL and ODBC external environments.

Here is an example of a function written in C++ that can be run within the database server or in an external environment.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
    )
{
    return TRUE;
}

// Note: extfn_use_new_api used only for
// execution in the database server

extern "C" __declspec( dllexport )
a_sql_uint32 extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}

extern "C" __declspec( dllexport )
void SimpleCFunction(
    an_extfn_api *api,
    void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;
    int *          intpnr;
    int            i, j, k;

    j = 1000;
    k = 0;
    for( i = 1; i <= 4; i++ )
    {
        result = api->get_value( arg_handle, i, &arg );
        if( result == 0 || arg.data == NULL ) break;
        if( arg.type & DT_TYPES != DT_INT ) break;
    }
}
```

```

        intptr = (int *) arg.data;
        k += *intptr * j;
        j = j / 10;
    }
    retval.type = DT_INT;
    retval.data = (void*)&k;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );
    api->set_value( arg_handle, 0, &retval, 0 );
    return;
}

```

When compiled into a dynamic link library or shared object, this function can be called from an external environment. An executable image called *dbexternc12* is started by the database server and this executable image loads the dynamic link library or shared object for you. Different versions of this executable are included with SQL Anywhere. For example, on Windows you may have both 32-bit and 64-bit executables.

Note that 32-bit or 64-bit versions of the database server can be used and either version can start 32-bit or 64-bit versions of *dbexternc12*. This is one of the advantages of using the external environment. Note that once *dbexternc12* is started by the database server, it does not terminate until the connection has been terminated or a STOP EXTERNAL ENVIRONMENT statement (with the correct environment name) is executed. Each connection that does an external environment call will get its own copy of *dbexternc12*.

To call the compiled native function, SimpleCFunction, a wrapper is defined as follows:

```

CREATE FUNCTION SimpleCDemo(
    IN arg1 INT,
    IN arg2 INT,
    IN arg3 INT,
    IN arg4 INT )
RETURNS INT
EXTERNAL NAME 'SimpleCFunction@c:\\c\\extdemo.dll'
LANGUAGE C_ODBC32;

```

This is almost identical to the way a compiled native function is described when it is to be loaded into the database server's address space. The one difference is the use of the LANGUAGE C\_ODBC32 clause. This clause indicates that SimpleCDemo is a function running in an external environment and that it is using 32-bit ODBC calls. The language specification of C\_ESQL32, C\_ESQL64, C\_ODBC32, or C\_ODBC64 tells the database server whether the external C function issues 32-bit or 64-bit ODBC, ESQL, or SQL Anywhere C API calls when making server-side requests.

When the native function uses none of the ODBC, ESQL, or SQL Anywhere C API calls to make server-side requests, then either C\_ODBC32 or C\_ESQL32 can be used for 32-bit applications and either C\_ODBC64 or C\_ESQL64 can be used for 64-bit applications. This is the case in the external C function shown above. It does not use any of these APIs.

To execute the sample compiled native function, execute the following statement.

```

SELECT SimpleCDemo(1,2,3,4);

```

To use server-side ODBC, the C/C++ code must use the default database connection. To get a handle to the database connection, call get\_value with an EXTFN\_CONNECTION\_HANDLE\_ARG\_NUM

argument. The argument tells the database server to return the current external environment connection rather than opening a new one.

```
#include <windows.h>
#include <stdio.h>
#include "odbc.h"
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
    )
{
    return TRUE;
}

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *api, void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;
    SQLRETURN      ret;

    ret = -1;
    // set up the return value struct
    retval.type = DT_INT;
    retval.data = (void*) &ret;
    retval.piece_len = retval.len.total_len =
        (a_sql_uint32) sizeof( int );

    result = api->get_value( arg_handle,
        EXTFN_CONNECTION_HANDLE_ARG_NUM,
        &arg );
    if( result == 0 || arg.data == NULL )
    {
        api->set_value( arg_handle, 0, &retval, 0 );
        return;
    }

    HDBC dbc = (HDBC)arg.data;
    HSTMT stmt = SQL_NULL_HSTMT;
    ret = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
    if( ret != SQL_SUCCESS ) return;
    ret = SQLExecDirect( stmt,
        (SQLCHAR *) "INSERT INTO odbcTab "
            "SELECT table_id, table_name "
            "FROM SYS.SYSTAB", SQL_NTS );
    if( ret == SQL_SUCCESS )
    {
        SQLExecDirect( stmt,
            (SQLCHAR *) "COMMIT", SQL_NTS );
    }
    SQLFreeHandle( SQL_HANDLE_STMT, stmt );

    api->set_value( arg_handle, 0, &retval, 0 );
    return;
}
```

If the above ODBC code is stored in the file *extodbc.cpp*, it can be built for Windows using the following commands (assuming that the SQL Anywhere software is installed in the folder *c:\sa12* and that Microsoft Visual C++ is installed).



```
cl extodbc.cpp /LD /Ic:\sa12\sdk\include odbc32.lib
```

The following example creates a table, defines the stored procedure wrapper to call the compiled native function, and then calls the native function to populate the table.

```
CREATE TABLE odbcTab(c1 int, c2 char(128));

CREATE FUNCTION ServerSideODBC( )
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extodbc.dll'
LANGUAGE C_ODBC32;

SELECT ServerSideODBC();

// The following statement should return two identical rows
SELECT COUNT(*) FROM odbcTab
UNION ALL
SELECT COUNT(*) FROM SYS.SYSTAB;
```

Similarly, to use server-side ESQL, the C/C++ code must use the default database connection. To get a handle to the database connection, call `get_value` with an `EXTFN_CONNECTION_HANDLE_ARG_NUM` argument. The argument tells the database server to return the current external environment connection rather than opening a new one.

```
#include <windows.h>
#include <stdio.h>

#include "sqlca.h"
#include "sqlda.h"
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
    )
{
    return TRUE;
}

EXEC SQL INCLUDE SQLCA;
static SQLCA *_sqlc;
EXEC SQL SET SQLCA "_sqlc";
EXEC SQL WHENEVER SQLERROR { ret = _sqlc->sqlcode; };

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *api, void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;

    EXEC SQL BEGIN DECLARE SECTION;
    char *stmt_text =
        "INSERT INTO esqlTab "
        "SELECT table_id, table_name "
        "FROM SYS.SYSTAB";
    char *stmt_commit =
        "COMMIT";
    EXEC SQL END DECLARE SECTION;

    int ret = -1;
```

```
// set up the return value struct
retval.type = DT_INT;
retval.data = (void*) &ret;
retval.piece_len = retval.len.total_len =
    (a_sql_uint32) sizeof( int );

result = api->get_value( arg_handle,
                       EXTFN_CONNECTION_HANDLE_ARG_NUM,
                       &arg );
if( result == 0 || arg.data == NULL )
{
    api->set_value( arg_handle, 0, &retval, 0 );
    return;
}
ret = 0;
_sqlc = (SQLCA *)arg.data;

EXEC SQL EXECUTE IMMEDIATE :stmt_text;
EXEC SQL EXECUTE IMMEDIATE :stmt_commit;

api->set_value( arg_handle, 0, &retval, 0 );
}
```

If the above embedded SQL statements are stored in the file *extesql.sqc*, it can be built for Windows using the following commands (assuming that the SQL Anywhere software is installed in the folder *c:\sa12* and that Microsoft Visual C++ is installed).

```
sqlpp extesql.sqc extesql.cpp
cl extesql.cpp /LD /Ic:\sa12\sdk\include c:\sa12\sdk\lib\x86\dblibtm.lib
```

The following example creates a table, defines the stored procedure wrapper to call the compiled native function, and then calls the native function to populate the table.

```
CREATE TABLE esqlTab(c1 int, c2 char(128));

CREATE FUNCTION ServerSideESQL( )
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extesql.dll'
LANGUAGE C_ESQL32;

SELECT ServerSideESQL();

// The following statement should return two identical rows
SELECT COUNT(*) FROM esqlTab
UNION ALL
SELECT COUNT(*) FROM SYS.SYSTAB;
```

As in the previous examples, to use server-side SQL Anywhere C API calls, the C/C++ code must use the default database connection. To get a handle to the database connection, call `get_value` with an `EXTFN_CONNECTION_HANDLE_ARG_NUM` argument. The argument tells the database server to return the current external environment connection rather than opening a new one. The following example shows the framework for obtaining the connection handle, initializing the C API environment, and transforming the connection handle into a connection object (**a `sqlany_connection`**) that can be used with the SQL Anywhere C API.

```
#include <windows.h>
#include "sacapidll.h"
#include "extfnapi.h"

BOOL APIENTRY DllMain( HMODULE hModule,
```

```

        DWORD   ul_reason_for_call,
        LPVOID  lpReserved
    )
    {
        return TRUE;
    }

extern "C" __declspec( dllexport )
void ServerSideFunction( an_extfn_api *extapi, void *arg_handle )
{
    short          result;
    an_extfn_value arg;
    an_extfn_value retval;
    unsigned       offset;
    char           *cmd;

    SQLAnywhereInterface  capi;
    a_sqlany_connection * sqlany_conn;
    unsigned int          max_api_ver;

    result = extapi->get_value( arg_handle,
                               EXTFN_CONNECTION_HANDLE_ARG_NUM,
                               &arg );

    if( result == 0 || arg.data == NULL )
    {
        return;
    }
    if( !sqlany_initialize_interface( &capi, NULL ) )
    {
        return;
    }
    if( !capi.sqlany_init( "MyApp",
                          SQLANY_CURRENT_API_VERSION,
                          &max_api_ver ) )
    {
        sqlany_finalize_interface( &capi );
        return;
    }
    sqlany_conn = sqlany_make_connection( arg.data );

    // processing code goes here

    capi.sqlany_fini();

    sqlany_finalize_interface( &capi );
    return;
}

```

If the above C code is stored in the file *extcapi.c*, it can be built for Windows using the following commands (assuming that the SQL Anywhere software is installed in the folder *c:\sa12* and that Microsoft Visual C++ is installed).

```

cl /LD /Tp extcapi.c /Tp c:\sa12\SDK\C\sacapidll.c
    /Ic:\sa12\SDK\Include c:\sa12\SDK\Lib\X86\dbcapi.lib

```

The following example defines the stored procedure wrapper to call the compiled native function, and then calls the native function.

```

CREATE FUNCTION ServerSideC()
RETURNS INT
EXTERNAL NAME 'ServerSideFunction@extcapi.dll'

```

```
LANGUAGE C_ESQL32;  
  
SELECT ServerSideC();
```

The LANGUAGE attribute in the above example specifies C\_ESQL32. For 64-bit applications, you would use C\_ESQL64. You must use the embedded SQL language attribute since the SQL Anywhere C API is built on the same layer (library) as ESQL.

As mentioned earlier, each connection that does an external environment call will start its own copy of *dbexternc12*. This executable application is loaded automatically by the server the first time an external environment call is made. However, you can use the START EXTERNAL ENVIRONMENT statement to preload *dbexternc12*. This is useful if you want to avoid the slight delay that is incurred when an external environment call is executed for the first time. Here is an example of the statement.

```
START EXTERNAL ENVIRONMENT C_ESQL32
```

Another case where preloading *dbexternc12* is useful is when you want to debug your external function. You can use the debugger to attach to the running *dbexternc12* process and set breakpoints in your external function.

The STOP EXTERNAL ENVIRONMENT statement is useful when updating a dynamic link library or shared object. It will terminate the native library loader, *dbexternc12*, for the current connection thereby releasing access to the dynamic link library or shared object. If multiple connections are using the same dynamic link library or shared object then each of their copies of *dbexternc12* must be terminated. The appropriate external environment name must be specified in the STOP EXTERNAL ENVIRONMENT statement. Here is an example of the statement.

```
STOP EXTERNAL ENVIRONMENT C_ESQL32
```

To return a result set from an external function, the compiled native function must use the native function call interface. This interface is fully described in [“SQL Anywhere external call interface” on page 563](#).

The following code fragment shows how to set up a result set information structure. It contains a column count, a pointer to an array of column information structures, and a pointer to an array of column data value structures. The example also uses the SQL Anywhere C API.

```
an_extfn_result_set_info    rs_info;  
  
int columns = capi.sqlany_num_cols( sqlany_stmt );  
  
an_extfn_result_set_column_info *col_info =  
    (an_extfn_result_set_column_info *)  
    malloc( columns * sizeof(an_extfn_result_set_column_info) );  
  
an_extfn_result_set_column_data *col_data =  
    (an_extfn_result_set_column_data *)  
    malloc( columns * sizeof(an_extfn_result_set_column_data) );  
  
rs_info.number_of_columns    = columns;  
rs_info.column_infos         = col_info;  
rs_info.column_data_values   = col_data;
```

The following code fragment shows how to describe the result set. It uses the SQL Anywhere C API to obtain column information for a SQL query that was executed previously by the C API. The information

that is obtained from the SQL Anywhere C API for each column is transformed into a column name, type, width, index, and null value indicator that will be used to describe the result set.

```

a_sqlany_column_info    info;
for( int i = 0; i < columns; i++ )
{
    if( sqlany_get_column_info( sqlany_stmt, i, &info ) )
    {
        // set up a column description
        col_info[i].column_name = info.name;
        col_info[i].column_type = info.native_type;
        switch( info.native_type )
        {
            case DT_DATE:          // DATE is converted to string by C API
            case DT_TIME:          // TIME is converted to string by C API
            case DT_TIMESTAMP:     // TIMESTAMP is converted to string by C API
            case DT_DECIMAL:       // DECIMAL is converted to string by C API
                col_info[i].column_type = DT_FIXCHAR;
                break;
            case DT_FLOAT:         // FLOAT is converted to double by C API
                col_info[i].column_type = DT_DOUBLE;
                break;
            case DT_BIT:          // BIT is converted to tinyint by C API
                col_info[i].column_type = DT_TINYINT;
                break;
        }
        col_info[i].column_width = info.max_size;
        col_info[i].column_index = i + 1; // column indices are origin 1
        col_info[i].column_can_be_null = info.nullable;
    }
}
// send the result set description
if( extapi->set_value( arg_handle,
                     EXTFN_RESULT_SET_ARG_NUM,
                     (an_extfn_value *)&rs_info,
                     EXTFN_RESULT_SET_DESCRIBE ) == 0 )
{
    // failed
    free( col_info );
    free( col_data );
    return;
}

```

Once the result set has been described, the result set rows can be returned. The following code fragment shows how to return the rows of the result set. It uses the SQL Anywhere C API to fetch the rows for a SQL query that was executed previously by the C API. The rows returned by the SQL Anywhere C API are sent back, one at a time, to the calling environment. The array of column data value structures must be filled in before returning each row. The column data value structure consists of a column index, a pointer to a data value, a data length, and an append flag.

```

a_sqlany_data_value *value = (a_sqlany_data_value *)
    malloc( columns * sizeof(a_sqlany_data_value) );

while( capi.sqlany_fetch_next( sqlany_stmt ) )
{
    for( int i = 0; i < columns; i++ )
    {
        if( capi.sqlany_get_column( sqlany_stmt, i, &value[i] ) )
        {
            col_data[i].column_index = i + 1;
            col_data[i].column_data = value[i].buffer;
        }
    }
}

```

```
        col_data[i].data_length = (a_sql_uint32)*(value[i].length);
        col_data[i].append = 0;
        if( *(value[i].is_null) )
        {
            // Received a NULL value
            col_data[i].column_data = NULL;
        }
    }
}
if( extapi->set_value( arg_handle,
                    EXTFN_RESULT_SET_ARG_NUM,
                    (an_extfn_value *)&rs_info,
                    EXTFN_RESULT_SET_NEW_ROW_FLUSH ) == 0 )
{
    // failed
    free( value );
    free( col_data );
    free( col_data );
    extapi->set_value( arg_handle, 0, &retval, 0 );
    return;
}
}
```

For more information about how to make server-side requests and how to return result sets from an external function, refer to the samples in *%SQLANY\SAMP12%\SQLAnywhere\ExternalEnvironments\ExternC*.

#### See also

- [“SQL Anywhere external call interface” on page 563](#)

## The Java external environment

SQL Anywhere includes support for Java stored procedures and functions. A Java stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM environment). It should be noted that there is one instance of the Java VM for each database rather than one instance per connection. Java stored procedures can return result sets.

There are a few prerequisites to using Java in the database support:

1. A copy of the Java Runtime Environment must be installed on the database server computer.
2. The SQL Anywhere database server must be able to locate the Java executable (the Java VM).

To use Java in the database, make sure that the database server is able to locate and start the Java executable. Verify that this can be done by executing:

```
START EXTERNAL ENVIRONMENT JAVA;
```

If the database server fails to start Java then the problem probably occurs because the database server is not able to locate the Java executable. In this case, you should execute an ALTER EXTERNAL

ENVIRONMENT statement to explicitly set the location of the Java executable. Make sure to include the executable file name.

```
ALTER EXTERNAL ENVIRONMENT JAVA
LOCATION 'java-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT JAVA
LOCATION 'c:\\jdk1.6.0\\jre\\bin\\java.exe';
```

You can query the location of the Java VM that the database server will use by executing the following SQL query:

```
SELECT db_property('JAVAVM');
```

Note that the START EXTERNAL ENVIRONMENT JAVA statement is not necessary other than to verify that the database server can start the Java VM. In general, making a Java stored procedure or function call starts the Java VM automatically.

Similarly, the STOP EXTERNAL ENVIRONMENT JAVA statement is not necessary to stop an instance of Java since the instance automatically goes away when the all connections to the database have terminated. However, if you are completely done with Java and you want to make it possible to free up some resources, then the STOP EXTERNAL ENVIRONMENT JAVA statement decrements the usage count for the Java VM.

Once you have verified that the database server can start the Java VM executable, the next thing to do is to install the necessary Java class code into the database. Do this by using the INSTALL JAVA statement. For example, you can execute the following statement to install a Java class from a file into the database.

```
INSTALL JAVA
NEW
FROM FILE 'java-class-file';
```

You can also install a Java JAR file into the database.

```
INSTALL JAVA
NEW
JAR 'jar-name'
FROM FILE 'jar-file';
```

Java classes can be installed from a variable, as follows:

```
CREATE VARIABLE JavaClass LONG VARCHAR;
SET JavaClass = xp_read_file('java-class-file')
INSTALL JAVA
NEW
FROM JavaClass;
```

Java JAR files can be installed from a variable, as follows:

```
CREATE VARIABLE JavaJar LONG VARCHAR;
SET JavaJar = xp_read_file('jar-file')
INSTALL JAVA
NEW
JAR 'jar-name'
FROM JavaJar;
```

To remove a Java class from the database, use the REMOVE JAVA statement, as follows:

```
REMOVE JAVA CLASS java-class
```

To remove a Java JAR from the database, use the REMOVE JAVA statement, as follows:

```
REMOVE JAVA JAR 'jar-name'
```

To modify existing Java classes, you can use the UPDATE clause of the INSTALL JAVA statement, as follows:

```
INSTALL JAVA  
UPDATE  
FROM FILE 'java-class-file'
```

You can also update existing Java JAR files in the database.

```
INSTALL JAVA  
UPDATE  
JAR 'jar-name'  
FROM FILE 'jar-file';
```

Java classes can be updated from a variable, as follows:

```
CREATE VARIABLE JavaClass LONG VARCHAR;  
SET JavaClass = xp_read_file('java-class-file')  
INSTALL JAVA  
UPDATE  
FROM JavaClass;
```

Java JAR files can be updated from a variable, as follows:

```
CREATE VARIABLE JavaJar LONG VARCHAR;  
SET JavaJar = xp_read_file('jar-file')  
INSTALL JAVA  
UPDATE  
FROM JavaJar;
```

Once the Java class is installed in the database, you can then create stored procedures and functions to interface to the Java methods. The EXTERNAL NAME string contains the information needed to call the Java method and to return OUT parameters and return values. The LANGUAGE attribute of the EXTERNAL NAME clause must specify JAVA. The format of the EXTERNAL NAME clause is:

EXTERNAL NAME '*java-call*' LANGUAGE JAVA

*java-call* :  
[*package-name*.]*class-name.method-name method-signature*

*method-signature* :  
( [ *field-descriptor*, ... ] ) *return-descriptor*

*field-descriptor* and *return-descriptor* :

```
Z  
|  
B  
|  
S  
|  
I  
|  
J
```



**F**  
**D**  
**C**  
**V**  
*[descriptor*  
**L***class-name;*

A Java method signature is a compact character representation of the types of the parameters and the type of the return value. If the number of parameters is less than the number indicated in the method-signature, then the difference must equal the number specified in DYNAMIC RESULT SETS, and each parameter in the method signature that is more than those in the procedure parameter list must have a method signature of `[Ljava/sql/ResultSet;`.

The *field-descriptor* and *return-descriptor* have the following meanings:

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
<i>L class-name;</i>	an instance of the class <i>class-name</i> . The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, <code>java/lang/String</code>
S	short
V	void
Z	Boolean
[	use one for each dimension of an array

For example:

```

double some_method(
    boolean a,
    int b,
    java.math.BigDecimal c,
    byte [][] d,
    java.sql.ResultSet[] rs ) {
}

```

would have the following signature:

```
'(ZILjava/math/BigDecimal;[[B[Ljava/sql/ResultSet;)D'
```

The following procedure creates an interface to a Java method. The Java method does not return any value (V).

```
CREATE PROCEDURE insertfix()  
EXTERNAL NAME 'JDBCEExample.InsertFixed()V'  
LANGUAGE JAVA;
```

The following procedure creates an interface to a Java method that has a String (Ljava/lang/String;) input argument. The Java method does not return any value (V).

```
CREATE PROCEDURE InvoiceMain( IN arg1 CHAR(50) )  
EXTERNAL NAME 'Invoice.main([Ljava/lang/String;)V'  
LANGUAGE JAVA;
```

The following procedure creates an interface to a Java method Invoice.init which takes a string argument (Ljava/lang/String;), a double (D), another string argument (Ljava/lang/String;), and another double (D), and returns no value (V).

```
CREATE PROCEDURE init( IN arg1 CHAR(50),  
                      IN arg2 DOUBLE,  
                      IN arg3 CHAR(50),  
                      IN arg4 DOUBLE)  
EXTERNAL NAME 'Invoice.init(Ljava/lang/String;DLjava/lang/String;D)V'  
LANGUAGE JAVA
```

The following Java example contains the function main which takes a string argument and writes it to the database server messages window. It also contains the function whoAreYou that returns a Java String.

```
import java.io.*;  
  
public class Hello  
{  
    public static void main( String[] args )  
    {  
        System.out.print( "Hello" );  
        for ( int i = 0; i < args.length; i++ )  
            System.out.print( " " + args[i] );  
        System.out.println();  
    }  
    public static String whoAreYou()  
    {  
        return( "I am SQL Anywhere." );  
    }  
}
```

The Java code above is placed in the file *Hello.java* and compiled using the Java compiler. The class file that results is loaded into the database as follows.

```
INSTALL JAVA  
NEW  
FROM FILE 'Hello.class';
```

Using Interactive SQL, the stored procedure that will interface to the method main in the class Hello is created as follows:

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )  
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'  
LANGUAGE JAVA;
```

Note that the argument to `main` is described as an array of `java.lang.String`. Using Interactive SQL, test the interface by executing the following SQL statement.

```
CALL HelloDemo('SQL Anywhere');
```

If you check the database server messages window, you will find the message written there. All output to `System.out` is redirected to the server messages window.

Using Interactive SQL, the function that will interface to the method `whoAreYou` in the class `Hello` is created as follows:

```
CREATE FUNCTION WhoAreYou()  
RETURNS LONG VARCHAR  
EXTERNAL NAME 'Hello.whoAreYou(V)Ljava/lang/String;'  
LANGUAGE JAVA;
```

Note that the function `whoAreYou` is described as returning a `java.lang.String`. Using Interactive SQL, test the interface by executing the following SQL statement.

```
SELECT WhoAreYou();
```

You should see the response in the Interactive SQL **Results** window.

In attempting to trouble shoot why a Java external environment did not start, that is, if the application gets a "main thread not found" error when a Java call is made, the DBA should check the following:

- If the Java VM is a different bitness than the database server, then ensure that the client libraries with the same bitness as the VM are installed on the database server computer.
- Ensure that the `sajdbc.jar` and `dbjdbc12/libdbjdbc12` shared objects are from the same software build.
- If more than one `sajdbc.jar` are on the database server computer, make sure they are all synchronized to the same software version.
- If the database server computer is very busy, then there is a chance the error is being reported due to a timeout.

#### See also

- [“Calling methods in the Java class” on page 391](#)
- [“Returning result sets from Java methods” on page 398](#)
- [“Java in the database” on page 385](#)

## The PERL external environment

SQL Anywhere includes support for Perl stored procedures and functions. A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance). It should be noted that there is a separate instance of the Perl executable for each connection that uses Perl stored procedures and functions. This behavior is

different from Java stored procedures and functions. For Java, there is one instance of the Java VM for each database rather than one instance per connection. The other major difference between Perl and Java is that Perl stored procedures do not return result sets, whereas Java stored procedures can return result sets.

There are a few prerequisites to using Perl in the database support:

1. Perl must be installed on the database server computer and the SQL Anywhere database server must be able to locate the Perl executable.
2. The DBD::SQLAnywhere driver must be installed on the database server computer.
3. On Windows, Microsoft Visual Studio must also be installed. This is a prerequisite since it is necessary for installing the DBD::SQLAnywhere driver.

For more information about installing the DBD::SQLAnywhere driver, see [“Perl DBI support” on page 611](#).

In addition to the above prerequisites, the database administrator must also install the SQL Anywhere Perl External Environment module. To install the external environment module:

### Install the external environment module (Windows)

- Run the following commands from the *SDK\PerlEnv* subdirectory of your SQL Anywhere installation:

```
perl Makefile.PL
nmake
nmake install
```

### Install the external environment module (Unix)

- Run the following commands from the *sdk/perlenv* subdirectory of your SQL Anywhere installation:

```
perl Makefile.PL
make
make install
```

Once the Perl external environment module has been built and installed, the Perl in the database support can be used. Note that Perl in the database support is only available with SQL Anywhere version 11 or later databases. If a SQL Anywhere 10 database is loaded, then an error indicating that external environments are not supported is returned when you try to use the Perl in the database support.

To use Perl in the database, make sure that the database server is able to locate and start the Perl executable. Verify that this can be done by executing:

```
START EXTERNAL ENVIRONMENT PERL;
```

If the database server fails to start Perl, then the problem probably occurs because the database server is not able to locate the Perl executable. In this case, you should execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the Perl executable. Make sure to include the executable file name.

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'perl-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'c:\\Perl\\bin\\perl.exe';
```

Note that the `START EXTERNAL ENVIRONMENT PERL` statement is not necessary other than to verify that the database server can start Perl. In general, making a Perl stored procedure or function call starts Perl automatically.

Similarly, the `STOP EXTERNAL ENVIRONMENT PERL` statement is not necessary to stop an instance of Perl since the instance automatically goes away when the connection terminates. However, if you are completely done with Perl and you want to free up some resources, then the `STOP EXTERNAL ENVIRONMENT PERL` statement releases the Perl instance for your connection.

Once you have verified that the database server can start the Perl executable, the next thing to do is to install the necessary Perl code into the database. Do this by using the `INSTALL` statement. For example, you can execute the following statement to install a Perl script from a file into the database.

```
INSTALL EXTERNAL OBJECT 'perl-script'
NEW
FROM FILE 'perl-file'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from an expression, as follows:

```
INSTALL EXTERNAL OBJECT 'perl-script'
NEW
FROM VALUE 'perl-statements'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from a variable, as follows:

```
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable = 'perl-statements';
INSTALL EXTERNAL OBJECT 'perl-script'
NEW
FROM VALUE PerlVariable
ENVIRONMENT PERL;
```

To remove Perl code from the database, use the `REMOVE` statement, as follows:

```
REMOVE EXTERNAL OBJECT 'perl-script'
```

To modify existing Perl code, you can use the `UPDATE` clause of the `INSTALL EXTERNAL OBJECT` statement, as follows:

```
INSTALL EXTERNAL OBJECT 'perl-script'
UPDATE
FROM FILE 'perl-file'
ENVIRONMENT PERL

INSTALL EXTERNAL OBJECT 'perl-script'
UPDATE
FROM VALUE 'perl-statements'
ENVIRONMENT PERL
```

```
SET PerlVariable = 'perl-statements';
INSTALL EXTERNAL OBJECT 'perl-script'
UPDATE
FROM VALUE PerlVariable
ENVIRONMENT PERL
```

Once the Perl code is installed in the database, you can then create the necessary Perl stored procedures and functions. When creating Perl stored procedures and functions, the LANGUAGE is always PERL and the EXTERNAL NAME string contains the information needed to call the Perl subroutines and to return OUT parameters and return values. The following global variables are available to the Perl code on each call:

- **\$sa\_perl\_return** This is used to set the return value for a function call.
- **\$sa\_perl\_argN** where N is a positive integer [0 .. n]. This is used for passing the SQL arguments down to the Perl code. For example, \$sa\_perl\_arg0 refers to argument 0, \$sa\_perl\_arg1 refers to argument 1, and so on.
- **\$sa\_perl\_default\_connection** This is used for making server-side Perl calls.
- **\$sa\_output\_handle** This is used for sending output from the Perl code to the database server messages window.

A Perl stored procedure can be created with any set of data types for input and output arguments, and for the return value. However, all non-binary data types are mapped to strings when making the Perl call while binary data is mapped to an array of numbers. A simple Perl example follows:

```
INSTALL EXTERNAL OBJECT 'SimplePerlExample'
NEW
FROM VALUE 'sub SimplePerlSub{
    return( ($_[0] * 1000) +
            ($_[1] * 100) +
            ($_[2] * 10) +
            $_[3] );
}'
ENVIRONMENT PERL;

CREATE FUNCTION SimplePerlDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
RETURNS INT
EXTERNAL NAME '<file=SimplePerlExample>
    $sa_perl_return = SimplePerlSub(
        $sa_perl_arg0,
        $sa_perl_arg1,
        $sa_perl_arg2,
        $sa_perl_arg3)'
LANGUAGE PERL;

// The number 1234 should appear
SELECT SimplePerlDemo(1,2,3,4);
```

The following Perl example takes a string and writes it to the database server messages window:

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
```

```

FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;

CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
  EXTERNAL NAME '<file=PerlConsoleExample>'
  WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;

// 'Hello world' should appear in the database server messages window
CALL PerlWriteToConsole( 'Hello world' );

```

To use server-side Perl, the Perl code must use the `$sa_perl_default_connection` variable. The following example creates a table and then calls a Perl stored procedure to populate the table:

```

CREATE TABLE perlTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePerlExample'
NEW
FROM VALUE 'sub ServerSidePerlSub
  { $sa_perl_default_connection->do(
    "INSERT INTO perlTab SELECT table_id, table_name FROM SYS.SYSTAB" );
    $sa_perl_default_connection->do(
      "COMMIT" );
  }'
ENVIRONMENT PERL;

CREATE PROCEDURE PerlPopulateTable()
  EXTERNAL NAME '<file=ServerSidePerlExample> ServerSidePerlSub()'
LANGUAGE PERL;

CALL PerlPopulateTable();

// The following should return 2 identical rows
SELECT count(*) FROM perlTab
UNION ALL
SELECT count(*) FROM SYS.SYSTAB;

```

For more information and examples on using the Perl in the database support, refer to the examples located in the `%SQLANYSAMPI2%\SQLAnywhere\ExternalEnvironments\Perl` directory.

## The PHP external environment

SQL Anywhere includes support for PHP stored procedures and functions. A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance). There is a separate instance of the PHP executable for each connection that uses PHP stored procedures and functions. This behavior is quite different from Java stored procedures and functions. For Java, there is one instance of the Java VM for each database rather than one instance per connection. The other major difference between PHP and Java is that PHP stored procedures do not return result sets, whereas Java stored procedures can return result sets. PHP only returns an object of type LONG VARCHAR, which is the output of the PHP script.

There are two prerequisites to using PHP in the database support:

1. A copy of PHP must be installed on the database server computer and SQL Anywhere database server must be able to locate the PHP executable.
2. The SQL Anywhere PHP extension (shipped with SQL Anywhere) must be installed on the database server computer. See [“PHP client deployment” on page 949](#).

In addition to the above two prerequisites, the database administrator must also install the SQL Anywhere PHP External Environment module. Prebuilt modules for several versions of PHP are included with the SQL Anywhere distribution. To install prebuilt modules, copy the appropriate driver module to your PHP extensions directory (which can be found in *php.ini*). On Unix, you can also use a symbolic link.

### Install the external environment module (Windows)

1. Locate the *php.ini* file for your PHP installation, and open it in a text editor. Locate the line that specifies the location of the **extension\_dir** directory. If **extension\_dir** is not set to any specific directory, it is a good idea to set it to point to an isolated directory for better system security.
2. Copy the desired external environment PHP module from the SQL Anywhere installation directory to your PHP extensions directory. Change the x.y to reflect the version you have selected.

```
copy "%SQLANY12%\Bin32\php-5.x.y_sqlanywhere_extenv12.dll"  
php-dir\ext
```

3. Add the following line to the Dynamic Extensions section of the *php.ini* file to load the external environment PHP module automatically. Change the x.y to reflect the version you have selected.

```
extension=php-5.x.y_sqlanywhere_extenv12.dll
```

Save and close *php.ini*.

4. Make sure that you have also installed the SQL Anywhere PHP driver from the SQL Anywhere installation directory into your PHP extensions directory. This file name follows the pattern *php-5.x.y\_sqlanywhere.dll* where x and y are the version numbers. It should match the version numbers of the file that you copied in step 2.

### Install the external environment module (Unix)

1. Locate the *php.ini* file for your PHP installation, and open it in a text editor. Locate the line that specifies the location of the **extension\_dir** directory. If **extension\_dir** is not set to any specific directory, it is a good idea to set it to point to an isolated directory for better system security.
2. Copy the desired external environment PHP module from the SQL Anywhere installation directory to your PHP installation directory. Change the x.y to reflect the version you have selected.

```
cp $SQLANY12/bin32/php-5.x.y_sqlanywhere_extenv12.so  
php-dir/ext
```

3. Add the following line to the Dynamic Extensions section of the *php.ini* file to load the external environment PHP module automatically. Change the x.y to reflect the version you have selected.

```
extension=php-5.x.y_sqlanywhere_extenv12.so
```

Save and close *php.ini*.



4. Make sure that you have also installed the SQL Anywhere PHP driver from the SQL Anywhere installation directory into your PHP extensions directory. This file name follows the pattern *php-5.x.y\_sqlanywhere.so* where x and y are the version numbers. It should match the version numbers of the file that you copied in step 2.

PHP in the database support is only available with SQL Anywhere version 11 or later databases. If a SQL Anywhere 10 database is loaded, then an error indicating that external environments are not supported is returned when you try to use the PHP in the database support.

To use PHP in the database, the database server must be able to locate and start the PHP executable. You can verify if the database server is able to locate and start the PHP executable by executing the following statement:

```
START EXTERNAL ENVIRONMENT PHP;
```

If you see a message that states that 'external executable' could not be found, then the problem is that the database server is not able to locate the PHP executable. In this case, you should execute an ALTER EXTERNAL ENVIRONMENT statement to explicitly set the location of the PHP executable including the executable name or you should ensure that the PATH environment variable includes the directory containing the PHP executable.

```
ALTER EXTERNAL ENVIRONMENT PHP
LOCATION 'php-path';
```

For example:

```
ALTER EXTERNAL ENVIRONMENT PHP
LOCATION 'c:\\php\\php-5.2.6-win32\\php.exe';
```

To restore the default setting, execute the following statement:

```
ALTER EXTERNAL ENVIRONMENT PHP
LOCATION 'php';
```

If you see a message that states that 'main thread' could not be found, then check for the following:

- Make sure that both the *php-5.x.y\_sqlanywhere* and *php-5.x.y\_sqlanywhere\_extenv12* modules are located in the directory indicated by **extension\_dir**. Check the installation steps described above.
- Make sure that both the *php-5.x.y\_sqlanywhere* and *php-5.x.y\_sqlanywhere\_extenv12* modules are listed in the Dynamic Extensions section of the *php.ini* file. Check the installation steps described above.
- Make sure that *phpenv.php* can be located. Check that the SQL Anywhere *bin32* folder is in your PATH.
- For Windows, make sure that the 32-bit DLLs (*dbcapi.dll*, *dblib12.dll*, *dbicu12.dll*, *dbicudt12.dll*, *dblgen12.dll*, and *dbextenv12.dll*) can be located. Check that the SQL Anywhere *bin32* folder is in your PATH.
- For Linux, Unix, and Mac OS X make sure that the 32-bit shared objects (*libdbcapi\_r*, *libdblib12\_r*, *libdbicu12\_r*, *libdbicudt12*, *dblgen12.res*, and *libdbextenv12\_r*) can be located. Check that the SQL Anywhere *bin32* folder is in your PATH.
- Make sure that the environment variable **PHPRC** is not set, or make sure that it points to the version of PHP that you intend to use.

The `START EXTERNAL ENVIRONMENT PHP` statement is not necessary other than to verify that the database server can start PHP. In general, making a PHP stored procedure or function call starts PHP automatically.

Similarly, the `STOP EXTERNAL ENVIRONMENT PHP` statement is not necessary to stop an instance of PHP since the instance automatically goes away when the connection terminates. However, if you are completely done with PHP and you want to free up some resources, then the `STOP EXTERNAL ENVIRONMENT PHP` statement releases the PHP instance for your connection.

Once you have verified that the database server can start the PHP executable, the next thing to do is to install the necessary PHP code into the database. Do this by using the `INSTALL` statement. For example, you can execute the following statement to install a particular PHP script into the database.

```
INSTALL EXTERNAL OBJECT 'php-script'  
NEW  
FROM FILE 'php-file'  
ENVIRONMENT PHP;
```

PHP code can also be built and installed from an expression as follows:

```
INSTALL EXTERNAL OBJECT 'php-script'  
NEW  
FROM VALUE 'php-statements'  
ENVIRONMENT PHP;
```

PHP code can also be built and installed from a variable as follows:

```
CREATE VARIABLE PHPVariable LONG VARCHAR;  
SET PHPVariable = 'php-statements';  
INSTALL EXTERNAL OBJECT 'php-script'  
NEW  
FROM VALUE PHPVariable  
ENVIRONMENT PHP;
```

To remove PHP code from the database, use the `REMOVE` statement as follows:

```
REMOVE EXTERNAL OBJECT 'php-script';
```

To modify existing PHP code, you can use the `UPDATE` clause of the `INSTALL` statement as follows:

```
INSTALL EXTERNAL OBJECT 'php-script'  
UPDATE  
FROM FILE 'php-file'  
ENVIRONMENT PHP;  
  
INSTALL EXTERNAL OBJECT 'php-script'  
UPDATE  
FROM VALUE 'php-statements'  
ENVIRONMENT PHP;  
  
SET PHPVariable = 'php-statements';  
INSTALL EXTERNAL OBJECT 'php-script'  
UPDATE  
FROM VALUE PHPVariable  
ENVIRONMENT PHP;
```

Once the PHP code is installed in the database, you can then go ahead and create the necessary PHP stored procedures and functions. When creating PHP stored procedures and functions, the `LANGUAGE`

is always PHP and the EXTERNAL NAME string contains the information needed to call the PHP subroutines and for returning OUT parameters.

The arguments are passed to the PHP script in the \$argv array, similar to the way PHP would take arguments from the command line (that is, \$argv[1] is the first argument). To set an output parameter, assign it to the appropriate \$argv element. The return value is always the output from the script (as a LONG VARCHAR).

A PHP stored procedure can be created with any set of data types for input or output arguments. However, the parameters are converted to and from a boolean, integer, double, or string for use inside the PHP script. The return value is always an object of type LONG VARCHAR. A simple PHP example follows:

```
INSTALL EXTERNAL OBJECT 'SimplePHPExample'
NEW
FROM VALUE '<?php function SimplePHPFunction(
    $arg1, $arg2, $arg3, $arg4 )
    { return ($arg1 * 1000) +
      ($arg2 * 100) +
      ($arg3 * 10) +
      $arg4;
    } ?>'
ENVIRONMENT PHP;

CREATE FUNCTION SimplePHPDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
RETURNS LONG VARCHAR
EXTERNAL NAME '<file=SimplePHPExample> print SimplePHPFunction(
    $argv[1], $argv[2], $argv[3], $argv[4]);'
LANGUAGE PHP;

// The number 1234 should appear
SELECT SimplePHPDemo(1,2,3,4);
```

For PHP, the EXTERNAL NAME string is specified in a single line of SQL.

To use server-side PHP, the PHP code can use the default database connection. To get a handle to the database connection, call sasql\_pconnect with an empty string argument (" or ""). The empty string argument tells the SQL Anywhere PHP driver to return the current external environment connection rather than opening a new one. The following example creates a table and then calls a PHP stored procedure to populate the table:

```
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<?php function ServerSidePHPSub() {
    $conn = sasql_pconnect( '' );
    sasql_query( $conn,
    "INSERT INTO phpTab
    SELECT table_id, table_name FROM SYS.SYSTAB" );
    sasql_commit( $conn );
} ?>'
ENVIRONMENT PHP;

CREATE PROCEDURE PHPPopulateTable()
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
```

```
LANGUAGE PHP;

CALL PHPPopulateTable();

// The following should return 2 identical rows
SELECT count(*) FROM phpTab
UNION ALL
SELECT count(*) FROM SYS.SYSTAB;
```

For PHP, the EXTERNAL NAME string is specified in a single line of SQL. In the above example, note that the single quotes are doubled-up because of the way quotes are parsed in SQL. If the PHP source code was in a file, then the single quotes would not be doubled-up.

To return an error back to the database server, throw a PHP exception. The following example shows how to do this.

```
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<?php function ServerSidePHPSub() {
    $conn = sasql_pconnect( '' );
    if( !sasql_query( $conn,
        "INSERT INTO phpTabNoExist
            SELECT table_id, table_name FROM SYS.SYSTAB" )
    ) throw new Exception(
        sasql_error( $conn ),
        sasql_errorcode( $conn )
    );
    sasql_commit( $conn );
} ?>'
ENVIRONMENT PHP;

CREATE PROCEDURE PHPPopulateTable()
EXTERNAL NAME
'<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;

CALL PHPPopulateTable();
```

The above example should terminate with error `SQLE_UNHANDLED_EXTENV_EXCEPTION` indicating that the table `phpTabNoExist` could not be found.

For more information and examples on using the PHP in the database support, refer to the examples located in the `%SQLANYSAMPI2%\SQLAnywhere\ExternalEnvironments\PHP` directory.

---

# Perl DBI support

DBD::SQLAnywhere is the SQL Anywhere database driver for DBI, which is a data access API for the Perl language. The DBI API specification defines a set of functions, variables and conventions that provide a consistent database interface independent of the actual database being used. Using DBI and DBD::SQLAnywhere, your Perl scripts have direct access to SQL Anywhere database servers.

## DBD::SQLAnywhere

DBD::SQLAnywhere is a driver for the Database Independent Interface for Perl (DBI) module written by Tim Bunce. Once you have installed the DBI module and DBD::SQLAnywhere, you can access and change the information in SQL Anywhere databases from Perl.

The DBD::SQLAnywhere driver is thread-safe when using Perl with `ithreads`.

### Requirements

The DBD::SQLAnywhere interface requires the following components.

- Perl 5.6.0 or later. On Windows, ActivePerl 5.6.0 build 616 or later is required.
- DBI 1.34 or later.
- A C compiler. On Windows, only the Microsoft Visual C++ compiler is supported.

## Installing Perl/DBI support on Windows

### Prepare your computer

1. Install ActivePerl 5.6.0 or later. You can use the ActivePerl installer to install Perl and configure your computer. You do not need to recompile Perl.
2. Install Microsoft Visual Studio and configure your environment.

If you did not choose to configure your environment at install time, you must set your `PATH`, `LIB`, and `INCLUDE` environment variables correctly before proceeding. Microsoft provides a batch file for this purpose. For 32-bit builds, a batch file called `vcvars32.bat` is included in the `vc\bin` subdirectory of the Visual Studio 2005 or 2008 installation. For 64-bit builds, look for a 64-bit version of this batch file such as `vcvarsamd64.bat`. Open a new system command prompt and run this batch file before continuing.

For more information about configuring a 64-bit Visual C++ build environment, see <http://msdn.microsoft.com/en-us/library/x4d2c09s.aspx>.

### Install the DBI Perl module on Windows

1. At a command prompt, change to the *bin* subdirectory of your ActivePerl installation directory.

The system command prompt is strongly recommended as the following steps may not work from alternative shells.

2. Using the Perl Module Manager, enter the following command.

```
ppm query dbi
```

If ppm fails to run, check that Perl is installed correctly.

This command should generate two lines of text similar to those shown below. In this case, the information indicates that ActivePerl version 5.8.1 build 807 is running and that DBI version 1.38 is installed.

```
Querying target 1 (ActivePerl 5.8.1.807)
  1. DBI [1.38] Database independent interface for Perl
```

Later versions of Perl may show instead a table similar to the following. In this case, the information indicates that DBI version 1.58 is installed.

name	version	abstract	area
DBI	1.58	Database independent interface for Perl	perl

If DBI is not installed, you must install it. To do so, enter the following command at the ppm prompt.

```
ppm install dbi
```

### Install DBD::SQLAnywhere on Windows

1. At a command prompt, change to the *SDK\Perl* subdirectory of your SQL Anywhere installation.
2. Enter the following commands to build and test DBD::SQLAnywhere.

```
perl Makefile.PL
```

```
nmake
```

If for any reason you need to start over, you can run the command **nmake clean** to remove any partially built targets.

3. To test DBD::SQLAnywhere, copy the sample database file to your *SDK\Perl* directory and make the tests.

```
copy "%SQLANYXSAMP12%\demo.db" .
```

```
dbeng12 demo
```

```
nmake test
```

If the tests do not run, ensure that the *bin32* or *bin64* subdirectory of the SQL Anywhere installation is in your path.

4. To complete the installation, execute the following command at the same prompt.

```
nmake install
```

The DBD::SQLAnywhere interface is now ready to use.

## Installing Perl/DBI support on Unix and Mac OS X

The following procedure documents how to install the DBD::SQLAnywhere interface on the supported Unix platforms, including Mac OS X.

### Prepare your computer

1. Install ActivePerl 5.6.0 build 616 or later.
2. Install a C compiler.

### Install the DBI Perl module on Unix and Mac OS X

1. Download the DBI module source from <http://www.cpan.org>.
2. Extract the contents of this file into a new directory.
3. At a command prompt, change to the new directory and execute the following commands to build the DBI module.

```
perl Makefile.PL  
make
```

If for any reason you need to start over, you can use the command **make clean** to remove any partially built targets.

4. Use the following command to test the DBI module.

```
make test
```

5. To complete the installation, execute the following command at the same prompt.

```
make install
```

6. Optionally, you can now delete the DBI source tree. It is no longer required.

### Install DBD::SQLAnywhere on Unix and Mac OS X

1. Make sure the environment is set up for SQL Anywhere.

Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from the SQL Anywhere installation directory:

In this shell...	Use this command...
sh, ksh, or bash	<code>. bin/sa_config.sh</code>
csh or tcsh	<code>source bin/sa_config.csh</code>

- At a shell prompt, change to the *sdk/perl* subdirectory of your SQL Anywhere installation.
- At a command prompt, run the following commands to build DBD::SQLAnywhere.

```
perl Makefile.PL
make
```

If for any reason you need to start over, you can use the command **make clean** to remove any partially built targets.

- To test DBD::SQLAnywhere, copy the sample database file to your *sdk/perl* directory and make the tests.

```
cp samples-dir/demo.db .
dbeng12 demo
make test
```

If the tests do not run, ensure that the *bin32* or *bin64* subdirectory of the SQL Anywhere installation is in your path.

- To complete the installation, execute the following command at the same prompt.

```
make install
```

The DBD::SQLAnywhere interface is now ready to use.

## Perl scripts that use DBD::SQLAnywhere

This section provides an overview of how to write Perl scripts that use the DBD::SQLAnywhere interface. DBD::SQLAnywhere is a driver for the DBI module. Complete documentation for the DBI module is available online at <http://dbi.perl.org>.

### Loading the DBI module

To use the DBD::SQLAnywhere interface from a Perl script, you must first tell Perl that you plan to use the DBI module. To do so, include the following line at the top of the file.

```
use DBI;
```



In addition, it is highly recommended that you run Perl in strict mode. This statement, which for example makes explicit variable definitions mandatory, is likely to greatly reduce the chance that you will run into mysterious errors due to such common mistakes as typographical errors.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

The DBI module automatically loads the DBD drivers, including DBD::SQLAnywhere, as required.

## Opening and closing a connection

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the connect method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the connect method are as follows:

1. "DBI:SQLAnywhere:" and additional connection parameters separated by semicolons.
2. A user name. Unless this string is blank, ";UID=*value*" is appended to the connection string.
3. A password value. Unless this string is blank, ";PWD=*value*" is appended to the connection string.
4. A pointer to a hash of default values. Settings such as AutoCommit, RaiseError, and PrintError may be set in this manner.

The following code sample opens and closes a connection to the SQL Anywhere sample database. You must start the database server and sample database before running this script.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my %defaults = (
    AutoCommit => 1, # Autocommit enabled.
    PrintError => 0 # Errors not automatically printed.
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$dbh->disconnect;
exit(0);
__END__
```

Optionally, you can append the user name or password value to the data-source string instead of supplying them as separate parameters. If you do so, supply a blank string for the corresponding argument. For example, in the above script may be altered by replacing the statement that opens the connections with these statements:

```
$data_src .= ";UID=$uid";
$data_src .= ";PWD=$pwd";
my $dbh = DBI->connect($data_src, '', '', \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
```

## Selecting data

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

SQL statements that return row sets must be prepared before being executed. The prepare method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of the result set.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
              FROM Customers
              ORDER BY GivenName, Surname";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
$db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
    $sth = $dbh->prepare($sel);
    $sth->execute;
    print "Fields:      $sth->{NUM_OF_FIELDS}\n";
    print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
    print join("\t\t", @{$sth->{NAME}}), "\n\n";
    while($row = $sth->fetchrow_arrayref) {
        print join("\t\t", @$row), "\n";
    }
    $sth = undef;
}
__END__
```

Prepared statements are not dropped from the database server until the Perl statement handle is destroyed. To destroy a statement handle, reuse the variable or set it to undef. Calling the finish method does not drop the handle. In fact, the finish method should not be called, except when you have decided not to finish reading a result set.

To detect handle leaks, the SQL Anywhere database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use `prepare_cached` sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the `max_cursor_count` and `max_statement_count` options.

### See also

- “`max_cursor_count` option” [[SQL Anywhere Server - Database Administration](#)]
- “`max_statement_count` option” [[SQL Anywhere Server - Database Administration](#)]

## How multiple result sets are processed

The method for handling multiple result sets from a query involves wrapping the fetch loop within another loop that moves between result sets.

SQL statements that return multiple result sets must be prepared before being executed. The `prepare` method returns a handle to the statement. You use the handle to execute the statement, then retrieve meta information about the result set and the rows of each of the result sets.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
               FROM Customers
               ORDER BY GivenName, Surname;
               SELECT *
               FROM Departments
               ORDER BY DepartmentID";

my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);

my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Cannot connect to $data_src: $DBI::errstr\n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
    my($sel, $dbh) = @_;
    my($row, $sth) = undef;
    $sth = $dbh->prepare($sel);
    $sth->execute;
    do {
        print "Fields:      $sth->{NUM_OF_FIELDS}\n";
        print "Params:      $sth->{NUM_OF_PARAMS}\n\n";
        print join("\t\t", @{$sth->{NAME}}), "\n\n";
    }
}
```

```
        while($row = $sth->fetchrow_arrayref) {
            print join("\t\t", @$row), "\n";
        }
        print "---end of results---\n\n";
    } while (defined $sth->more_results);
    $sth = undef;
}
__END__
```

## Inserting rows

Inserting rows requires a handle to an open connection. The simplest method is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the execute method.

The following sample program inserts two new customers. Although the row values appear as literal strings, you may want to read the values from a file.

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $ins_stmt = "INSERT INTO Customers (ID, GivenName, Surname,
        Street, City, State, Country, PostalCode,
        Phone, CompanyName)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
my %defaults = (
    AutoCommit => 0, # Require explicit commit or rollback.
    PrintError => 0
);
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
    or die "Can't connect to $data_src: $DBI::errstr\n";
$dbh->insert($ins_stmt, $dbh);
$dbh->commit;
$dbh->disconnect;
exit(0);

sub db_insert {
    my($ins, $dbh) = @_;
    my($sth) = undef;
    my @rows = (
        "801,Alex,Alt,5 Blue Ave,New York,NY,USA,10012,5185553434,BXM",
        "802,Zach,Zed,82 Fair St,New York,NY,USA,10033,5185552234,Zap"
    );
    $sth = $dbh->prepare($ins);
    my $row = undef;
    foreach $row ( @rows ) {
        my @values = split(/,/, $row);
        $sth->execute(@values);
    }
}
__END__
```

---

# Python support

The SQL Anywhere Python database interface, `sqlanydb`, is a data access API for the Python language. This section describes how to use SQL Anywhere with Python.

## sqlanydb

The SQL Anywhere Python database interface, `sqlanydb`, is a data access API for the Python language. The Python Database API specification defines a set of methods that provides a consistent database interface independent of the actual database being used. Using the `sqlanydb` module, your Python scripts have direct access to SQL Anywhere database servers.

The `sqlanydb` module implements, with extensions, the Python Database API specification v2.0 written by Marc-André Lemburg. Once you have installed the `sqlanydb` module, you can access and change the information in SQL Anywhere databases from Python.

For information about the Python Database API specification v2.0, see <http://www.python.org/dev/peps/pep-0249/>.

The `sqlanydb` module is thread-safe when using Python with threads.

### Requirements

The `sqlanydb` module requires the following components.

- Python is required. For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
- The `ctypes` module is required. To test if the `ctypes` module is present, open a command prompt window and run Python.

At the Python prompt, enter the following statement.

```
import ctypes
```

If you see an error message, then `ctypes` is not present. The following is an example.

```
>>> import ctypes
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named ctypes
```

If `ctypes` is not included in your Python installation, install it. Installs can be found in the SourceForge.net files section at [http://sourceforge.net/project/showfiles.php?group\\_id=71702](http://sourceforge.net/project/showfiles.php?group_id=71702).

Peak EasyInstall also installs `ctypes`. To download Peak EasyInstall, go to <http://peak.telecommunity.com/DevCenter/EasyInstall>.

## Installing Python support on Windows

### Prepare your computer

1. Install Python. For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
2. Install the ctypes module if missing.

### Install the sqlanydb module on Windows

1. At a system command prompt, change to the *SDK\Python* subdirectory of your SQL Anywhere installation.
2. Run the following command to install sqlanydb.

```
python setup.py install
```

3. To test sqlanydb, copy the sample database file to your *SDK\Python* directory and run a test.

```
copy "%SQLANY%SAMP12%\demo.db" .  
dbeng12 demo  
python Scripts\test.py
```

If the tests do not run, ensure that the *bin32* or *bin64* subdirectory of the SQL Anywhere installation is in your path.

The sqlanydb module is now ready to use.

## Installing Python support on Unix and Mac OS X

The following procedure documents how to install the sqlanydb module on the supported Unix platforms, including Mac OS X.

### Prepare your computer

1. Install Python. For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
2. Install the ctypes module if missing.

### Install the sqlanydb module on Unix and Mac OS X

1. Make sure the environment is set up for SQL Anywhere.

Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from the SQL Anywhere installation directory (*bin64* may be used in place of *bin32*, if you have the 64-bit software installed):

In this shell...	Use this command...
sh, ksh, or bash	<code>. bin32/sa_config.sh</code>
csh or tcsh	<code>source bin32/sa_config.csh</code>

- At a shell prompt, change to the *sdk/python* subdirectory of your SQL Anywhere installation.
- Enter the following command to install sqlanydb.

```
python setup.py install
```

- To test sqlanydb, copy the sample database file to your *sdk/python* directory and run a test.

```
cp samples-dir/demo.db .
dbeng12 demo
python scripts/test.py
```

The test script makes a connection to the database server and executes a SQL query. If successful, the test displays the message `sqlanydb successfully installed`.

If the test does not run, ensure that the *bin32* or *bin64* subdirectory of the SQL Anywhere installation is in your path.

The sqlanydb module is now ready to use.

## Python scripts that use sqlanydb

This section provides an overview of how to write Python scripts that use the sqlanydb interface. Complete documentation for the API is available online at <http://www.python.org/dev/peps/pep-0249/>.

### Loading the sqlanydb module

To use the sqlanydb module from a Python script, you must first load it by including the following line at the top of the file.

```
import sqlanydb
```

### Opening and closing a connection

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the `connect` method. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The parameters to the `connect` method are specified as a series of `keyword=value` pairs delimited by commas.

```
sqlanydb.connect( keyword=value, ...)
```

Some common connection parameters are as follows:

- **DataSourceName="dsn"** A short form for this connection parameter is **DSN="dsn"**. An example is DataSourceName="SQL Anywhere 12 Demo".
- **UserID="user-id"** A short form for this connection parameter is **UID="user-id"**. An example is UserID="DBA".
- **Password="passwd"** A short form for this connection parameter is **PWD="passwd"**. An example is Password="sql".
- **DatabaseFile="db-file"** A short form for this connection parameter is **DBF="db-file"**. An example is DatabaseFile="demo.db".

The following code sample opens and closes a connection to the SQL Anywhere sample database. You must start the database server and sample database before running this script.

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( userid="DBA",
                       password="sql" )

# Close the connection
con.close()
```

To avoid starting the database server manually, you could use a data source that is configured to start the server. This is shown in the following example.

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( DSN="SQL Anywhere 12 Demo" )

# Close the connection
con.close()
```

### See also

- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## Selecting data

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

The cursor method is used to create a cursor on the open connection. The execute method is used to create a result set. The fetchall method is used to obtain the rows in this result set.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA",
```



```

        password="sql" )
    cursor = con.cursor()

    # Execute a SQL string
    sql = "SELECT * FROM Employees"
    cursor.execute(sql)

    # Get a cursor description which contains column names
    desc = cursor.description
    print len(desc)

    # Fetch all results from the cursor into a sequence,
    # display the values as column name=value pairs,
    # and then close the connection
    rowset = cursor.fetchall()
    for row in rowset:
        for col in range(len(desc)):
            print "%s=%s" % (desc[col][0], row[col] )
        print
    cursor.close()
    con.close()

```

## Inserting rows

The simplest way to insert rows into a table is to use a non-parameterized INSERT statement, meaning that values are specified as part of the SQL statement. A new statement is constructed and executed for each new row. As in the previous example, a cursor is required to execute SQL statements.

The following sample program inserts two new customers into the sample database. Before disconnecting, it commits the transactions to the database.

```

import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA", pwd="sql" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801, 'Alex', 'Alt', '5 Blue Ave', 'New York', 'NY',
         'USA', '10012', '5185553434', 'BXM'),
        (802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY',
         'USA', '10033', '5185552234', 'Zap'))

# Set up a SQL INSERT
parms = ("%s", " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql % rows[0]
cursor.execute(sql % rows[0])
print sql % rows[1]
cursor.execute(sql % rows[1])
cursor.close()
con.commit()
con.close()

```

An alternate technique is to use a parameterized INSERT statement, meaning that question marks are used as placeholders for values. The executemany method is used to execute an INSERT statement for each

member of the set of rows. The new row values are supplied as a single argument to the `executemany` method.

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA", pwd="sql" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
        'USA','10012','5185553434','BXM'),
        (802,'Zach','Zed','82 Fair St','New York','NY',
        'USA','10033','5185552234','Zap'))

# Set up a parameterized SQL INSERT
parms = ("?", " " * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql
cursor.executemany(sql, rows)
cursor.close()
con.commit()
con.close()
```

Although both examples may appear to be equally suitable techniques for inserting row data into a table, the latter example is superior for a couple of reasons. If the data values are obtained by prompts for input, then the first example is susceptible to injection of rogue data including SQL statements. In the first example, the `execute` method is called for each row to be inserted into the table. In the second example, the `executemany` method is called only once to insert all the rows into the table.

## Database type conversion

To control how database types are mapped into Python objects when results are fetched from the database server, conversion callbacks can be registered.

Callbacks are registered using the module level `register_converter` method. This method is called with the database type as the first parameter and the conversion function as the second parameter. For example, to request that `sqlanydb` create `Decimal` objects for data in any column described as having type `DT_DECIMAL`, you would use the following example:

```
import sqlanydb
import decimal

def convert_to_decimal(num):
    return decimal.Decimal(num)

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_decimal)
```

Converters may be registered for the following database types:

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
```

```
DT_LONGVARCHAR
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
DT_UNSMALLINT
DT_UNSSMALLINT
DT_UNSBIGINT
DT_BIT
```

The following example demonstrates how to convert decimal results to integer resulting in the truncation of any digits after the decimal point. Note that the salary amount displayed when the application is run is an integral value.

```
import sqlanydb

def convert_to_int(num):
    return int(float(num))

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="DBA",
                       password="sql" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

---

---

# PHP support

## SQL Anywhere PHP extension

PHP, which stands for **PHP: Hypertext Preprocessor**, is an open source scripting language. Although it can be used as a general-purpose scripting language, it was designed to be a convenient language in which to write scripts that could be embedded with HTML documents. Unlike scripts written in JavaScript, which are frequently executed by the client, PHP scripts are processed by the web server, and the resulting HTML output sent to the clients. The syntax of PHP is derived from that of other popular languages, such as Java and Perl.

To make it a convenient language in which to develop dynamic web pages, PHP provides the ability to retrieve information from many popular databases, such as SQL Anywhere. Included with SQL Anywhere is an extension that provides access to SQL Anywhere databases from PHP. You can use the SQL Anywhere PHP extension and the PHP language to write standalone scripts and create dynamic web pages that rely on information stored in SQL Anywhere databases.

The SQL Anywhere PHP extension provides a native means of accessing your databases from PHP. You might prefer it to other PHP data access techniques because it is simple, and it helps to avoid system resource leaks that can occur with other techniques.

Prebuilt versions of the PHP extension are provided for Windows, Linux, and Solaris and are installed in the binaries subdirectories of your SQL Anywhere installation. Source code for the SQL Anywhere PHP extension is installed in the *sdk\php* subdirectory of your SQL Anywhere installation.

For more information and the latest SQL Anywhere PHP drivers, see <http://www.sybase.com/detail?id=1019698>.

For information about installing the SQL Anywhere PHP extension, see “[PHP client deployment](#)” on page 949.

## Testing the PHP extension

Once all of the required PHP components have been installed on your system, you can do a quick check to verify that the SQL Anywhere PHP extension is working correctly.

### Test the PHP extension

1. Make sure that the *bin32* subdirectory of your SQL Anywhere installation is in your path. The SQL Anywhere PHP extension requires the *bin32* directory to be in your path.
2. At a command prompt, run the following command to start the SQL Anywhere sample database.

```
dbeng12 "%SQLANYSAMP12%\demo.db"
```

The command starts a database server using the sample database.

3. At a command prompt, change to the *SDK\PHP\Examples* subdirectory of your SQL Anywhere installation. Make sure that the **php** executable directory is included in your path. Enter the following command:

```
php test.php
```

Messages similar to the following should appear. If the PHP command is not recognized, verify that PHP is in your path.

```
Installation successful
Using php-5.2.11_sqlanywhere.dll
Connected successfully
```

If the SQL Anywhere PHP extension does not load, you can use the command "php -i" for helpful information about your PHP setup. Search for **extension\_dir** and **sqlanywhere** in the output from this command.

4. When you are done, stop the SQL Anywhere database server by clicking **Shut Down** in the database server messages window.

#### See also

- [“Creating PHP test pages” on page 628](#)

## Running PHP test scripts in your web pages

This section describes how to write PHP test scripts that query the sample database and display information about PHP.

### Creating PHP test pages

The following instructions apply to all configurations.

To test whether PHP is set up properly, the following procedure describes how to create and run a web page that calls `phpinfo`. The PHP function, `phpinfo`, generates a page of system setup information. The output tells you whether PHP is working properly.

For information about installing PHP, see <http://us2.php.net/install>.

#### Create a PHP information test page

1. Create a file in your root web content directory named *info.php*.

If you are not sure which directory to use, check your web server's configuration file. In Apache installations, the content directory is often called *htdocs*. If you are using Mac OS X, the web content directory name may depend on which account you are using:

- If you are the System Administrator on a Mac OS X system, use */Library/WebServer/Documents*.
- If you are a Mac OS X user, place the file in */Users/your-user-name/Sites/*.

2. Insert the following code into this file:

```
<?php phpinfo(); ?>
```

This confirms that your installation of PHP and your web server are working together properly.

3. At a command prompt, run the following command to start the SQL Anywhere sample database (if you have not already done so):

```
dbeng12 "%SQLANYAMP12%\demo.db"
```

4. To test that PHP and your web server are working correctly with SQL Anywhere:
  - a. Copy the file *connect.php* from the *sdk\php\examples* directory to your root web content directory.
  - b. From a web browser, access the *connect.php* page.  
The message `Connected successfully` should appear.

### Create the query page that uses the SQL Anywhere PHP extension

- Create a file containing the following PHP code in your root web content directory named *sa\_test.php*.

```
<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" );
$result = sasql_query( $conn, "SELECT * FROM Employees" );
sasql_result_all( $result );
sasql_free_result( $result );
sasql_disconnect( $conn );
?>
```

## Accessing your test web pages

The following procedure describes how to view your test pages from a web browser, after installing and configuring PHP and the SQL Anywhere PHP extension.

### View your web pages

1. Restart your web server.

For example, to start the Apache web server, run the following command from the *bin* subdirectory of your Apache installation:

```
apachectl start
```

2. On Linux or Mac OS X, set the SQL Anywhere environment variables using one of the supplied scripts.

Depending on which shell you are using, enter the appropriate command to source the SQL Anywhere configuration script from your SQL Anywhere installation directory:

In this shell ...	... use this command
sh, ksh, or bash	<code>. /bin32/sa_config.sh</code>
csh or tcsh	<code>source /bin32/sa_config.csh</code>

3. Start the SQL Anywhere database server.

For example, to access the test web pages described above, use the following command to start the SQL Anywhere sample database.

```
dbeng12 "%SQLANYSAMP12%\demo.db"
```

4. To access the test pages from a browser that is running on the same computer as the server, enter the following URLs:

For this test page...	Use this URL...
<i>info.php</i>	<code>http://localhost/info.php</code>
<i>sa_test.php</i>	<code>http://localhost/sa_test.php</code>

If everything is configured correctly, the *sa\_test* page displays the contents of the Employees table.

## Writing PHP scripts

This section describes how to write PHP scripts that use the SQL Anywhere PHP extension to access SQL Anywhere databases.

The source code for these and other examples is located in the *SDK\PHP\Examples* subdirectory of your SQL Anywhere installation.

## Database connections

To make a connection to a database, pass a standard SQL Anywhere connection string to the database server as a parameter to the `sasql_connect` function. The `<?php` and `?>` tags tell the web server that it should let PHP execute the code that lies between them and replace it with the PHP output.

The source code for this example is contained in your SQL Anywhere installation in a file called *connect.php*.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    echo "Connection failed\n";
} else {
    echo "Connected successfully\n";
}
```



```

        sasql_close( $conn );
    }?>

```

This script attempts to make a connection to a database on a local server. For this code to succeed, the SQL Anywhere sample database or one with identical credentials must be started on a local server.

## Retrieving data from a database

One use of PHP scripts in web pages is to retrieve and display information contained in a database. The following examples demonstrate some useful techniques.

### Simple select query

The following PHP code demonstrates a convenient way to include the result set of a SELECT statement in a web page. This sample is designed to connect to the SQL Anywhere sample database and return a list of customers.

This code can be embedded in a web page, provided your web server is configured to execute PHP scripts.

The source code for this sample is contained in your SQL Anywhere installation in a file called *query.php*.

```

<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    echo "sasql_connect failed\n";
} else {
    echo "Connected successfully\n";
    # Execute a SELECT statement
    $result = sasql_query( $conn, "SELECT * FROM Customers" );
    if( ! $result ) {
        echo "sasql_query failed!";
    } else {
        echo "query completed successfully\n";
        # Generate HTML from the result set
        sasql_result_all( $result );
        sasql_free_result( $result );
    }
    sasql_close( $conn );
}
?>

```

The `sasql_result_all` function fetches all the rows of the result set and generates an HTML output table to display them. The `sasql_free_result` function releases the resources used to store the result set.

### Fetching by column name

In certain cases, you may not want to display all the data from a result set, or you may want to display the data in a different manner. The following sample illustrates how you can exercise greater control over the output format of the result set. PHP allows you to display as much information as you want in whatever manner you choose.

The source code for this sample is contained in your SQL Anywhere installation in a file called *fetch.php*.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( ! $conn ) {
    die ("Connection failed");
} else {
    # Connected successfully.
}
# Execute a SELECT statement
$result = sasql_query( $conn, "SELECT * FROM Customers" );
if( ! $result ) {
    echo "sasql_query failed!";
    return 0;
} else {
    echo "query completed successfully\n";
}
# Retrieve meta information about the results
$num_cols = sasql_num_fields( $result );
$num_rows = sasql_num_rows( $result );
echo "Num of rows = $num_rows\n";
echo "Num of cols = $num_cols\n";
while( ($field = sasql_fetch_field( $result )) ) {
    echo "Field # : $field->id \n";
    echo "\tname      : $field->name \n";
    echo "\tlength   : $field->length \n";
    echo "\ttype     : $field->type \n";
}
# Fetch all the rows
$curr_row = 0;
while( ($row = sasql_fetch_row( $result )) ) {
    $curr_row++;
    $curr_col = 0;
    while( $curr_col < $num_cols ) {
        echo "$row[$curr_col]\t|";
        $curr_col++;
    }
    echo "\n";
}
# Clean up.
sasql_free_result( $result );
sasql_disconnect( $conn );
?>
```

The `sasql_fetch_array` function returns a single row from the table. The data can be retrieved by column names and column indexes.

The `sasql_fetch_assoc` function returns a single row from the table as an associative array. The data can be retrieved by using the column names as indexes. The following is an example.

```
<?php
# Connect using the default user ID and password
$conn = sasql_connect("UID=DBA;PWD=sql");

/* check connection */
if( sasql_errorcode() ) {
    printf("Connect failed: %s\n", sasql_error());
    exit();
}

$query = "SELECT Surname, Phone FROM Employees ORDER by EmployeeID";

if( $result = sasql_query($conn, $query) ) {
```

```

    /* fetch associative array */
    while( $row = sasql_fetch_assoc($result) ) {
        printf ("%s (%s)\n", $row["Surname"], $row["Phone"]);
    }

    /* free result set */
    sasql_free_result($result);
}

/* close connection */
sasql_close($conn);
?>

```

Two other similar methods are provided in the PHP interface: `sasql_fetch_row` returns a row that can be searched by column indexes only, while `sasql_fetch_object` returns a row that can be searched by column names only.

For an example of the `sasql_fetch_object` function, see the *fetch\_object.php* example script.

### Nested result sets

When a `SELECT` statement is sent to the database, a result set is returned. The `sasql_fetch_row` and `sasql_fetch_array` functions retrieve data from the individual rows of a result set, returning each row as an array of columns that can be queried further.

The source code for this sample is contained in your SQL Anywhere installation in a file called *nested.php*.

```

<?php
$conn = sasql_connect( "UID=DBA;PWD=sql" );
if( $conn ) {
    // get the GROUPO user id
    $result = sasql_query( $conn,
        "SELECT user_id FROM SYS.SYSUSER " .
        "WHERE user_name='GROUPO'" );
    if( $result ) {
        $row = sasql_fetch_array( $result );
        $user = $row[0];
    } else {
        $user = 0;
    }
    // get the tables created by user GROUPO
    $result = sasql_query( $conn,
        "SELECT table_id, table_name FROM SYS.SYSTABLE " .
        "WHERE creator = $user" );
    if( $result ) {
        $num_rows = sasql_num_rows( $result );
        echo "Returned rows : $num_rows\n";
        while( $row = sasql_fetch_array( $result ) ) {
            echo "Table: $row[1]\n";
            $query = "SELECT table_id, column_name FROM SYS.SYSCOLUMN " .
                "WHERE table_id = '$row[table_id]'";
            $result2 = sasql_query( $conn, $query );
            if( $result2 ) {
                echo "Columns:";
                while( $detailed = sasql_fetch_array( $result2 ) ) {
                    echo " $detailed[column_name]";
                }
                sasql_free_result( $result2 );
            }
        }
    }
}

```

```
        echo "\n\n";
    }
    sasql_free_result( $result );
}
sasql_disconnect( $conn );
}
?>
```

In the above sample, the SQL statement selects the table ID and name for each table from SYSTAB. The `sasql_query` function returns an array of rows. The script iterates through the rows using the `sasql_fetch_array` function to retrieve the rows from an array. An inner iteration goes through the columns of each row and prints their values.

## Web forms

PHP can take user input from a web form, pass it to the database server as a SQL query, and display the result that is returned. The following example demonstrates a simple web form that gives the user the ability to query the sample database using SQL statements and display the results in an HTML table.

The source code for this sample is contained in your SQL Anywhere installation in a file called *webisql.php*.

```
<?php
echo "<HTML>\n";
$name = $_POST["qname"];
$name = str_replace( "\\\"", "", $name );
echo "<form method=post action=webisql.php>\n";
echo "<br>Query: <input type=text Size=80 name=qname value=\"\$name\">\n";
echo "<input type=submit>\n";
echo "</form>\n";
echo "<HR><br>\n";
if( ! $name ) {
    echo "No Current Query\n";
    return;
}
# Connect to the database
$con_str = "UID=DBA;PWD=sql;SERVER=demo;LINKS=tcPIP";
$conn = sasql_connect( $con_str );
if( ! $conn ) {
    echo "sasql_connect failed\n";
    echo "</html>\n";
    return 0;
}
$name = str_replace( "\\\"", "", $name );
$result = sasql_query( $conn, $name );
if( ! $result ) {
    echo "sasql_query failed!";
} else {
    // echo "query completed successfully\n";
    sasql_result_all( $result, "border=1" );
    sasql_free_result( $result );
}
sasql_disconnect( $conn );
echo "</html>\n";
?>
```

This design could be extended to handle complex web forms by formulating customized SQL queries based on the values entered by the user.

## BLOBs in PHP applications

SQL Anywhere databases can store any type of data as a binary large object (BLOB). If that data is of a type readable by a web browser, a PHP script can easily retrieve it from the database and display it on a dynamically generated page.

BLOB fields are often used for storing non-text data, such as images in GIF or JPG format. Numerous types of data can be passed to a web browser without any need for third-party software or data type conversion. The following sample illustrates the process of adding an image to the database and then retrieving it again to be displayed in a web browser.

This sample is similar to the sample code in the files *image\_insert.php* and *image\_retrieve.php* of your SQL Anywhere installation. These samples also illustrate the use of a BLOB column for storing images.

```
<?php
    $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die("Cannot connect to database");
    $create_table = "CREATE TABLE images (ID INTEGER PRIMARY KEY, img IMAGE)";
    sasql_query( $conn, $create_table);
    $insert = "INSERT INTO images VALUES (99,
xp_read_file('ianywhere_logo.gif'))";
    sasql_query( $conn, $insert );
    $query = "SELECT img FROM images WHERE ID = 99";
    $result = sasql_query($conn, $query);
    $data = sasql_fetch_row($result);
    $img = $data[0];
    header("Content-type: image/gif");
    echo $img;
    sasql_disconnect($conn);
?>
```

To be able to send the binary data from the database directly to a web browser, the script must set the data's MIME type using the header function. In this case, the browser is told to expect a GIF image so it can display it correctly.

## How to build the SQL Anywhere PHP extension on Unix and Mac OS X

To connect PHP to SQL Anywhere using the SQL Anywhere PHP extension on Unix and Mac OS X, you must add the SQL Anywhere PHP extension's files to PHP's source tree, and then re-compile PHP.

### Requirements

The following is a list of software you need to have on your system to complete to use the SQL Anywhere PHP extension on Unix and Mac OS X:

- A SQL Anywhere installation, which can run on the same computer as the Apache web server, or on a different computer.
- The source code for the SQL Anywhere PHP extension, which can be downloaded from [http://download.sybase.com/ianywhere/php/2.0.3/src/sasql\\_php.zip](http://download.sybase.com/ianywhere/php/2.0.3/src/sasql_php.zip).

You also need **sqlpp** and *libdblib12.so* (Unix) or *libdblib12.dylib* (Mac OS X) installed (check your SQL Anywhere *lib32* directory).

- The PHP source code, which can be downloaded from <http://www.php.net>.  
Version 5.2.11 and 5.3.2 of PHP are recent stable releases. For a list of supported versions, see <http://www.sybase.com/detail?id=1068981>.
- The Apache web server source code, which can be downloaded from <http://httpd.apache.org>.  
If you are going to use a pre-built version of Apache, make sure that you have **apache** and **apache-devel** installed.
- If you plan to use the Unified ODBC PHP extension, you need to have *libdbodbc12.so* (Unix) or *libdbodbc12.dylib* (Mac OS X) installed (check your SQL Anywhere *lib32* directory).

The following binaries should be installed from your Unix installation disk if they are not already installed, and can be found as RPMs:

- make
- automake
- autoconf
- libtool (glibtool for Mac OS X)
- makeinfo
- bison
- gcc
- cpp
- glibc-devel
- kernel-headers
- flex

You must have the same access privileges as the person who installed PHP to perform certain steps of the installation. Most Unix-based systems offer a **sudo** command that allows users with insufficient permissions to execute certain commands as a user with the right to execute them.

## Adding the SQL Anywhere PHP extension files to the PHP source tree

1. Download the SQL Anywhere PHP extension source code from <http://www.sybase.com/detail?id=1019698>. Look for the section entitled **Building the Driver from Source**.
2. From the directory where you saved the SQL Anywhere PHP extension, extract the files to the *ext* subdirectory of the PHP source tree (Mac OS X users should replace **tar** with **gntar**):

```
$ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
```

For example:

```
$ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.11/ext
```

3. Make PHP aware of the extension:

```
$ cd PHP-source-directory/ext/sqlanywhere
$ touch *
$ cd ~/PHP-source-directory
$ ./buildconf
```

The following example is for PHP version 5.2.11. You must change php-5.2.11 below to the version of PHP you are using.

```
$ cd ~/php-5.2.11/ext/sqlanywhere
$ touch *
$ cd ~/php-5.2.11
$ ./buildconf
```

4. Verify that PHP is aware of the extension:

```
$ ./configure -help | egrep sqlanywhere
```

If you were successful in making PHP aware of the SQL Anywhere extension, you should see the following text:

```
--with-sqlanywhere=[DIR]
```

If you are unsuccessful, keep track of the output of this command and post it to the SQL Anywhere Forum at <http://sqlanywhere-forum.sybase.com/> for assistance.

## Compiling Apache and PHP

PHP can be compiled as a shared module of a web server (such as Apache) or as a CGI executable. If you are using a web server that is not supported by PHP, or if you want to execute PHP scripts in your command shell rather than on a web page, you should compile PHP as a CGI executable. Otherwise, if you want to install PHP to operate in conjunction with Apache, compile it as an Apache module.

### See also

- “Compiling PHP as an Apache module” on page 637
- “Compiling PHP as a CGI executable” on page 640

## Compiling PHP as an Apache module

### Compile PHP as an Apache module

The first two steps configure Apache so that it recognizes shared modules. If you have a compiled version of Apache already installed on your system, proceed to step 3. Note that Mac OS X comes with a pre-installed Apache web server.

1. Configure Apache to recognize shared modules.

Execute the following command (entered all on one line) from the directory where your Apache files were extracted:

```
$ cd Apache-source-directory
$ ./configure --enabled-shared=max --enable-module=most --
prefix=Apache-installation-directory
```

The following example is for Apache version 2.2.9. You must change **apache\_2.2.9** to the version of Apache you are using.

```
$ cd ~/apache_2.2.9
$ ./configure --enabled-shared=max --enable-module=most --
prefix=/usr/local/web/apache
```

2. Recompile and install the relevant components:

```
$ make
$ make install
```

Now you are ready to compile PHP to operate as an Apache module.

3. Make sure the environment is set up for SQL Anywhere.

Depending on which shell you are using, enter the appropriate command from the directory where SQL Anywhere is installed (by default, this is */opt/sqlanywhere12*). On Mac OS X, the default directory is */Applications/SQLAnywhere12/System*.

If you are using this shell...	...use this command
sh, ksh, bash	<code>. ./bin32/sa_config.sh</code>
csh, tcsh	<code>source ./bin32/sa_config.csh</code>

4. Configure PHP as an Apache module to include the SQL Anywhere PHP extension.

Execute the following commands:

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere --with-apxs=/Apache-installation-
directory/bin/apxs
```

The following example is for PHP version 5.2.11. You must change *php-5.2.11* to the version of PHP you are using.

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere --with-apxs=/usr/local/web/apache/bin/
apxs
```

The *configure* script will try to determine the version and location of your SQL Anywhere installation. In the output from the command, you should see lines similar to the following:

```
checking for SQL Anywhere support... yes
checking SQL Anywhere install dir... /opt/sqlanywhere12
checking SQL Anywhere version... 12
```

5. Recompile the relevant components:

```
$ make
```

6. Check that the libraries are properly linked.

- Linux users (the following example assumes you are using PHP version 5):



```
ldd ../libs/libphp5.so
```

- Mac OS X users:

Refer to your *httpd.conf* configuration file to determine where *libphp5.so* is on your computer. Perform the check with the following command:

```
otool -L $LIBPHP5_DIR/libphp5.so
```

*\$LIBPHP5\_DIR* is the directory where *libphp5.so* is located, according to your server configuration.

This command outputs a list of the libraries that *libphp5.so* uses. Verify that *libdblib12.so* is in the list.

7. Install the PHP binaries in Apache's *lib* directory:

```
$ make install
```

8. Perform verification. PHP does this automatically. All you need is to make sure that your *httpd.conf* configuration file is verified so that Apache will recognize *.php* files as PHP scripts.

*httpd.conf* is stored in the *conf* subdirectory of the *Apache* directory:

```
$ cd Apache-installation-directory/conf
```

For example:

```
$ cd /usr/local/web/apache/conf
```

Make a backup copy of *httpd.conf* before editing the file (you can replace **pico** with the text editor of your choice):

```
$ cp httpd.conf httpd.conf.backup
$ pico httpd.conf
```

Add or uncomment the following lines in *httpd.conf* (they are not located together in the file):

```
LoadModule php5_module    libexec/libphp5.so
AddModule mod_php5.c
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

#### Note

On Mac OS X, the last two lines should be added or uncommented in *httpd\_macosxserver.conf*.

The first two lines point Apache to the files that are used for interpreting PHP code, while the other two lines declare file types for files whose extension is *.php* or *.phps* so Apache can recognize and deal with them appropriately.

#### See also

- [“Running PHP test scripts in your web pages” on page 628](#)

## Compiling PHP as a CGI executable

### Compile PHP as a CGI executable

1. Make sure the environment is set up for SQL Anywhere.

For instructions on setting up the environment for SQL Anywhere, follow the instructions in step 4 of [“Compiling PHP as an Apache module” on page 637](#).

2. Configure PHP as a CGI executable and with the SQL Anywhere PHP extension.

Execute the following command from the directory where your PHP files were extracted:

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere
```

For example:

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere
```

The configuration script will try to determine the version and location of your SQL Anywhere installation. If you examine the output of this command, you should see lines similar to the following:

```
checking for SQL Anywhere support... yes
checking      SQL Anywhere install dir... /opt/sqlanywhere12
checking      SQL Anywhere version... 12
```

3. Compile the executable:

```
$ make
```

4. Install the components.

```
$ make install
```

### See also

- [“Running PHP test scripts in your web pages” on page 628](#)

## SQL Anywhere PHP API reference

The PHP API supports the following functions:

**Connections**

- “sasql\_close”
- “sasql\_connect”
- “sasql\_disconnect”
- “sasql\_error”
- “sasql\_errorcode”
- “sasql\_insert\_id”
- “sasql\_message”
- “sasql\_pconnect”
- “sasql\_set\_option”

**Queries**

- “sasql\_affected\_rows”
- “sasql\_next\_result”
- “sasql\_query”
- “sasql\_real\_query”
- “sasql\_store\_result”
- “sasql\_use\_result”

**Result sets**

- “sasql\_data\_seek”
- “sasql\_fetch\_array”
- “sasql\_fetch\_assoc”
- “sasql\_fetch\_field”
- “sasql\_fetch\_object”
- “sasql\_fetch\_row”
- “sasql\_field\_count”
- “sasql\_free\_result”
- “sasql\_num\_rows”
- “sasql\_result\_all”

**Transactions**

- “sasql\_commit”
- “sasql\_rollback”

## Statements

- `“sasql_prepare”`
- `“sasql_stmt_affected_rows”`
- `“sasql_stmt_bind_param”`
- `“sasql_stmt_bind_param_ex”`
- `“sasql_stmt_bind_result”`
- `“sasql_stmt_close”`
- `“sasql_stmt_data_seek”`
- `“sasql_stmt_execute”`
- `“sasql_stmt_fetch”`
- `“sasql_stmt_field_count”`
- `“sasql_stmt_free_result”`
- `“sasql_stmt_insert_id”`
- `“sasql_stmt_next_result”`
- `“sasql_stmt_num_rows”`
- `“sasql_stmt_param_count”`
- `“sasql_stmt_reset”`
- `“sasql_stmt_result_metadata”`
- `“sasql_stmt_send_long_data”`
- `“sasql_stmt_store_result”`

## Miscellaneous

- `“sasql_escape_string”`
- `“sasql_get_client_info”`

# **sasql\_affected\_rows**

## Prototype

int **sasql\_affected\_rows**( sasql\_conn *\$conn* )

## Description

Returns the number of rows affected by the last SQL statement. This function is typically used for INSERT, UPDATE, or DELETE statements. For SELECT statements, use the `sasql_num_rows` function.

## Parameters

**\$conn** The connection resource returned by a connect function.

## Returns

The number of rows affected.

## See also

- `“sasql_num_rows”`

## sasql\_commit

### Prototype

```
bool sasql_commit( sasql_conn $conn )
```

### Description

Ends a transaction on the SQL Anywhere database and makes any changes made during the transaction permanent. Useful only when the auto\_commit option is Off.

### Parameters

**\$conn** The connection resource returned by a connect function.

### Returns

TRUE on success or FALSE on failure.

### See also

- [“sasql\\_rollback”](#)
- [“sasql\\_set\\_option”](#)
- [“sasql\\_pconnect”](#)
- [“sasql\\_disconnect”](#)

## sasql\_close

### Prototype

```
bool sasql_close( sasql_conn $conn )
```

### Description

Closes a previously opened database connection.

### Parameters

**\$conn** The connection resource returned by a connect function.

### Returns

TRUE on success or FALSE on failure.

## sasql\_connect

### Prototype

```
sasql_conn sasql_connect( string $con_str )
```

## Description

Establishes a connection to a SQL Anywhere database.

## Parameters

**\$con\_str** A connection string as recognized by SQL Anywhere.

## Returns

A positive SQL Anywhere connection resource on success, or an error and 0 on failure.

## See also

- [“sasql\\_pconnect”](#)
- [“sasql\\_disconnect”](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“SQL Anywhere database connections” \[SQL Anywhere Server - Database Administration\]](#)

# sasql\_data\_seek

## Prototype

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

## Description

Positions the cursor on row *row\_num* on the *\$result* that was opened using *sasql\_query*.

## Parameters

**\$result** The result resource returned by the *sasql\_query* function.

**row\_num** An integer that represents the new position of the cursor within the result resource. For example, specify 0 to move the cursor to the first row of the result set or 5 to move it to the sixth row. Negative numbers represent rows relative to the end of the result set. For example, -1 moves the cursor to the last row in the result set and -2 moves it to the second-last row.

## Returns

TRUE on success or FALSE on error.

## See also

- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_assoc”](#)
- [“sasql\\_fetch\\_row”](#)
- [“sasql\\_fetch\\_object”](#)
- [“sasql\\_query”](#)

## **sasql\_disconnect**

### **Prototype**

bool **sasql\_disconnect**( sasql\_conn \$conn )

### **Description**

Closes a connection that has been opened with `sasql_connect` or `sasql_pconnect`.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

### **Returns**

TRUE on success or FALSE on error.

### **See also**

- [“sasql\\_connect”](#)
- [“sasql\\_pconnect”](#)

## **sasql\_error**

### **Prototype**

string **sasql\_error**( [ sasql\_conn \$conn ] )

### **Description**

Returns the error text of the most recently executed SQL Anywhere PHP function. Error messages are stored per connection. If no `$conn` is specified, then `sasql_error` returns the last error message where no connection was available. For example, if you call `sasql_connect` and the connection fails, then call `sasql_error` with no parameter for `$conn` to get the error message. If you want to obtain the corresponding SQL Anywhere error code value, use the `sasql_errorcode` function.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

### **Returns**

A string describing the error. For a list of error messages, see [“SQL Anywhere error messages”](#) [*Error Messages*].

### **See also**

- [“sasql\\_errorcode”](#)
- [“sasql\\_sqlstate”](#)
- [“sasql\\_set\\_option”](#)
- [“sasql\\_stmt\\_erno”](#)
- [“sasql\\_stmt\\_error”](#)

## sasql\_errorcode

### Prototype

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

### Description

Returns the error code of the most-recently executed SQL Anywhere PHP function. Error codes are stored per connection. If no *\$conn* is specified, then `sasql_errorcode` returns the last error code where no connection was available. For example, if you are calling `sasql_connect` and the connection fails, then call `sasql_errorcode` with no parameter for the *\$conn* to get the error code. If you want to get the corresponding error message use the `sasql_error` function.

### Parameters

**\$conn** The connection resource returned by a connect function.

### Returns

An integer representing a SQL Anywhere error code. An error code of 0 means success. A positive error code indicates success with warnings. A negative error code indicates failure. For a list of error codes, see [“SQL Anywhere error messages sorted by SQLCODE” \[Error Messages\]](#).

### See also

- `“sasql_connect”`
- `“sasql_pconnect”`
- `“sasql_error”`
- `“sasql_sqlstate”`
- `“sasql_set_option”`
- `“sasql_stmt_errno”`
- `“sasql_stmt_error”`

## sasql\_escape\_string

### Prototype

```
string sasql_escape_string( sasql_conn $conn, string $str )
```

### Description

Escapes all special characters in the supplied string. The special characters that are escaped are `\r`, `\n`, `'`, `"`, `;`, `\`, and the NULL character. This function is an alias of `sasql_real_escape_string`.

### Parameters

**\$conn** The connection resource returned by a connect function.

**\$string** The string to be escaped.



**Returns**

The escaped string.

**See also**

- [“sasql\\_real\\_escape\\_string”](#)
- [“sasql\\_connect”](#)

## sasql\_fetch\_array

**Prototype**

```
array sasql_fetch_array( sasql_result $result [, int $result_type ])
```

**Description**

Fetches one row from the result set. This row is returned as an array that can be indexed by the column names or by the column indexes.

**Parameters**

**\$result** The result resource returned by the `sasql_query` function.

**\$result\_type** This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `SASQL_ASSOC`, `SASQL_NUM`, or `SASQL_BOTH`. It defaults to `SASQL_BOTH`.

By using the `SASQL_ASSOC` constant this function will behave identically to the `sasql_fetch_assoc` function, while `SASQL_NUM` will behave identically to the `sasql_fetch_row` function. The final option `SASQL_BOTH` will create a single array with the attributes of both.

**Returns**

An array that represents a row from the result set, or `FALSE` when no rows are available.

**See also**

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_assoc”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_row”](#)
- [“sasql\\_fetch\\_object”](#)

## sasql\_fetch\_assoc

**Prototype**

```
array sasql_fetch_assoc( sasql_result $result )
```

## Description

Fetches one row from the result set as an associative array.

## Parameters

**\$result** The result resource returned by the `sasql_query` function.

## Returns

An associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or `FALSE` if there are no more rows in resultset.

## See also

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_row”](#)
- [“sasql\\_fetch\\_object”](#)

# sasql\_fetch\_field

## Prototype

```
object sasql_fetch_field( sasql_result $result [, int $field_offset ] )
```

## Description

Returns an object that contains information about a specific column.

## Parameters

**\$result** The result resource returned by the `sasql_query` function.

**\$field\_offset** An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

## Returns

An object that has the following properties:

- **id** contains the field's number.
- **name** contains the field's name.
- **numeric** indicates whether the field is a numeric value.
- **length** returns the field's native storage size.
- **type** returns the field's type.

- **native\_type** returns the field's native type. These are values like DT\_FIXCHAR, DT\_DECIMAL or DT\_DATE. See [“Embedded SQL data types” on page 449](#).
- **precision** returns the field's numeric precision. This property is only set for fields with native\_type equal to DT\_DECIMAL.
- **scale** returns the field's numeric scale. This property is only set for fields with native\_type equal to DT\_DECIMAL.

**See also**

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_assoc”](#)
- [“sasql\\_fetch\\_row”](#)
- [“sasql\\_fetch\\_object”](#)

## sasql\_fetch\_object

**Prototype**

object **sasql\_fetch\_object**( sasql\_result *\$result* )

**Description**

Fetches one row from the result set as an object.

**Parameters**

**\$result** The result resource returned by the sasql\_query function.

**Returns**

An object representing the fetched row in the result set where each property name matches one of the result set column names, or FALSE if there are no more rows in result set.

**See also**

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_assoc”](#)
- [“sasql\\_fetch\\_row”](#)

## sasql\_fetch\_row

**Prototype**

array **sasql\_fetch\_row**( sasql\_result *\$result* )

### Description

Fetches one row from the result set. This row is returned as an array that can be indexed by the column indexes only.

### Parameters

**\$result** The result resource returned by the `sasql_query` function.

### Returns

An array that represents a row from the result set, or `FALSE` when no rows are available.

### See also

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_assoc”](#)
- [“sasql\\_fetch\\_object”](#)

## sasql\_field\_count

### Prototype

```
int sasql_field_count( sasql_conn $conn )
```

### Description

Returns the number of columns (fields) the last result contains.

### Parameters

**\$conn** The connection resource returned by a connect function.

### Returns

A positive number of columns, or `FALSE` if `$conn` is not valid.

## sasql\_field\_seek

### Prototype

```
bool sasql_field_seek( sasql_result $result, int $field_offset )
```

### Description

Sets the field cursor to the given offset. The next call to `sasql_fetch_field` will retrieve the field definition of the column associated with that offset.

**Parameters**

**\$result** The result resource returned by the `sasql_query` function.

**\$field\_offset** An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

**Returns**

TRUE on success or FALSE on error.

## **sasql\_free\_result**

**Prototype**

```
bool sasql_free_result( sasql_result $result )
```

**Description**

Frees database resources associated with a result resource returned from `sasql_query`.

**Parameters**

**\$result** The result resource returned by the `sasql_query` function.

**Returns**

TRUE on success or FALSE on error.

**See also**

- [“sasql\\_query”](#)

## **sasql\_get\_client\_info**

**Prototype**

```
string sasql_get_client_info( )
```

**Description**

Returns the version information of the client.

**Parameters**

None

**Returns**

A string that represents the SQL Anywhere client software version. The returned string is of the form X.Y.Z.W where X is the major version number, Y is the minor version number, Z is the patch number, and W is the build number (for example, 10.0.1.3616).

## **sasql\_insert\_id**

### **Prototype**

int **sasql\_insert\_id**( sasql\_conn \$conn )

### **Description**

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

The sasql\_insert\_id function is provided for compatibility with MySQL databases.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

### **Returns**

The ID generated for an AUTOINCREMENT column by a previous INSERT statement or zero if last insert did not affect an AUTOINCREMENT column. The function can return FALSE if the \$conn is not valid.

## **sasql\_message**

### **Prototype**

bool **sasql\_message**( sasql\_conn \$conn, string \$message )

### **Description**

Writes a message to the server messages window.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

**\$message** A message to be written to the server messages window.

### **Returns**

TRUE on success or FALSE on failure.

## **sasql\_multi\_query**

### **Prototype**

bool **sasql\_multi\_query**( sasql\_conn \$conn, string \$sql\_str )

## Description

Prepares and executes one or more SQL queries specified by *\$sql\_str* using the supplied connection resource. Each query is separated from the other using semicolons.

The first query result can be retrieved or stored using `sasql_use_result` or `sasql_store_result`. `sasql_field_count` can be used to check if the query returns a result set or not.

All subsequent query results can be processed using `sasql_next_result` and `sasql_use_result/ sasql_store_result`.

## Parameters

**\$conn** The connection resource returned by a connect function.

**\$sql\_str** One or more SQL statements separated by semicolons.

For more information about SQL statements, see “[SQL statements](#)” [*SQL Anywhere Server - SQL Reference*].

## Returns

TRUE on success or FALSE on failure.

## See also

- [“sasql\\_store\\_result”](#)
- [“sasql\\_use\\_result”](#)
- [“sasql\\_field\\_count”](#)

# sasql\_next\_result

## Prototype

```
bool sasql_next_result( sasql_conn $conn )
```

## Description

Prepares the next result set from the last query that executed on *\$conn*.

## Parameters

**\$conn** The connection resource returned by a connect function.

## Returns

FALSE if there is no other result set to be retrieved. TRUE if there is another result to be retrieved. Call `sasql_use_result` or `sasql_store_result` to retrieve the next result set.

## See also

- [“sasql\\_use\\_result”](#)
- [“sasql\\_store\\_result”](#)

## sasql\_num\_fields

### Prototype

```
int sasql_num_fields( sasql_result $result )
```

### Description

Returns the number of fields that a row in the *\$result* contains.

### Parameters

**\$result** The result resource returned by the `sasql_query` function.

### Returns

Returns the number of fields in the specified result set.

### See also

- [“sasql\\_num\\_rows”](#)
- [“sasql\\_query”](#)

## sasql\_num\_rows

### Prototype

```
int sasql_num_rows( sasql_result $result )
```

### Description

Returns the number of rows that the *\$result* contains.

### Parameters

**\$result** The result resource returned by the `sasql_query` function.

### Returns

A positive number if the number of rows is exact, or a negative number if it is an estimate. To get the exact number of rows, the database option `row_counts` must be set permanently on the database, or temporarily on the connection. See [“sasql\\_set\\_option” on page 659](#).

### See also

- [“sasql\\_num\\_fields”](#)
- [“sasql\\_query”](#)



## sasql\_pconnect

### Prototype

```
sasql_conn sasql_pconnect( string $con_str )
```

### Description

Establishes a persistent connection to a SQL Anywhere database. Because of the way Apache creates child processes, you may observe a performance gain when using `sasql_pconnect` instead of `sasql_connect`. Persistent connections may provide improved performance in a similar fashion to connection pooling. If your database server has a limited number of connections (for example, the personal database server is limited to 10 concurrent connections), caution should be exercised when using persistent connections. Persistent connections could be attached to each of the child processes, and if you have more child processes in Apache than there are available connections, you will receive connection errors.

### Parameters

**\$con\_str** A connection string as recognized by SQL Anywhere.

### Returns

A positive SQL Anywhere persistent connection resource on success, or an error and 0 on failure.

### See also

- [“sasql\\_connect”](#)
- [“sasql\\_disconnect”](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“SQL Anywhere database connections” \[SQL Anywhere Server - Database Administration\]](#)

## sasql\_prepare

### Prototype

```
sasql_stmt sasql_prepare( sasql_conn $conn, string $sql_str )
```

### Description

Prepares the supplied SQL string.

### Parameters

**\$conn** The connection resource returned by a connect function.

**\$sql\_str** The SQL statement to be prepared. The string can include parameter markers by embedding question marks at the appropriate positions.

For more information about SQL statements, see [“SQL statements” \[SQL Anywhere Server - SQL Reference\]](#).

## Returns

A statement object or FALSE on failure.

## See also

- [“sasql\\_stmt\\_param\\_count”](#)
- [“sasql\\_stmt\\_bind\\_param”](#)
- [“sasql\\_stmt\\_bind\\_param\\_ex”](#)
- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_execute”](#)
- [“sasql\\_connect”](#)
- [“sasql\\_pconnect”](#)

# sasql\_query

## Prototype

mixed **sasql\_query**( sasql\_conn \$conn, string \$sql\_str [, int \$result\_mode ] )

## Description

Prepares and executes the SQL query *\$sql\_str* on the connection identified by *\$conn* that has already been opened using `sasql_connect` or `sasql_pconnect`.

The `sasql_query` function is equivalent to calling two functions, `sasql_real_query` and one of `sasql_store_result` or `sasql_use_result`.

## Parameters

**\$conn** The connection resource returned by a connect function.

**\$sql\_str** A SQL statement supported by SQL Anywhere.

**\$result\_mode** Either `SASQL_USE_RESULT`, or `SASQL_STORE_RESULT` (the default).

For more information about SQL statements, see [“SQL statements” \[SQL Anywhere Server - SQL Reference\]](#).

## Returns

FALSE on failure; TRUE on success for INSERT, UPDATE, DELETE, CREATE; `sasql_result` for SELECT.

## See also

- [“sasql\\_real\\_query”](#)
- [“sasql\\_free\\_result”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_object”](#)
- [“sasql\\_fetch\\_row”](#)

## **sasql\_real\_escape\_string**

### **Prototype**

string **sasql\_real\_escape\_string**( sasql\_conn *\$conn*, string *\$str* )

### **Description**

Escapes all special characters in the supplied string. The special characters that are escaped are \r, \n, ', ", :, \, and the NULL character.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

**\$string** The string to be escaped.

### **Returns**

The escaped string or FALSE on error.

### **See also**

- [“sasql\\_escape\\_string”](#)
- [“sasql\\_connect”](#)

## **sasql\_real\_query**

### **Prototype**

bool **sasql\_real\_query**( sasql\_conn *\$conn*, string *\$sql\_str* )

### **Description**

Executes a query against the database using the supplied connection resource. The query result can be retrieved or stored using `sasql_store_result` or `sasql_use_result`. The `sasql_field_count` function can be used to check if the query returns a result set or not.

Note that the `sasql_query` function is equivalent to calling this function and one of `sasql_store_result` or `sasql_use_result`.

### **Parameters**

**\$conn** The connection resource returned by a connect function.

**\$sql\_str** A SQL statement supported by SQL Anywhere.

### **Returns**

TRUE on success or FALSE on failure.

**See also**

- [“sasql\\_query”](#)
- [“sasql\\_store\\_result”](#)
- [“sasql\\_use\\_result”](#)
- [“sasql\\_field\\_count”](#)

## sasql\_result\_all

**Prototype**

```
bool sasql_result_all( resource $result
[, $html_table_format_string
[, $html_table_header_format_string
[, $html_table_row_format_string
[, $html_table_cell_format_string
]]]])
```

**Description**

Fetches all results of the *\$result* and generates an HTML output table with an optional formatting string.

**Parameters**

**\$result** The result resource returned by the `sasql_query` function.

**\$html\_table\_format\_string** A format string that applies to HTML tables. For example, "**Border=1; Cellpadding=5**". The special value `none` does not create an HTML table. This is useful if you want to customize your column names or scripts. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**\$html\_table\_header\_format\_string** A format string that applies to column headings for HTML tables. For example, "**bgcolor=#FF9533**". The special value `none` does not create an HTML table. This is useful if you want to customize your column names or scripts. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**\$html\_table\_row\_format\_string** A format string that applies to rows within HTML tables. For example, "**onclick='alert('this')'**". If you would like different formats that alternate, use the special token `><`. The left side of the token indicates which format to use on odd rows and the right side of the token is used to format even rows. If you do not place this token in your format string, all rows have the same format. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**\$html\_table\_cell\_format\_string** A format string that applies to cells within HTML table rows. For example, "**onclick='alert('this')'**". If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**Returns**

TRUE on success or FALSE on failure.

**See also**

- [“sasql\\_query”](#)

## sasql\_rollback

**Prototype**

```
bool sasql_rollback( sasql_conn $conn )
```

**Description**

Ends a transaction on the SQL Anywhere database and discards any changes made during the transaction. This function is only useful when the `auto_commit` option is Off.

**Parameters**

**\$conn** The connection resource returned by a connect function.

**Returns**

TRUE on success or FALSE on failure.

**See also**

- [“sasql\\_commit”](#)
- [“sasql\\_set\\_option”](#)

## sasql\_set\_option

**Prototype**

```
bool sasql_set_option( sasql_conn $conn, string $option, mixed $value )
```

**Description**

Sets the value of the specified option on the specified connection. You can set the value for the following options:

Name	Description	Default
<code>auto_commit</code>	When this option is set to on, the database server commits after executing each statement.	on
<code>row_counts</code>	When this option is set to FALSE, the <code>sasql_num_rows</code> function returns an estimate of the number of rows affected. If you want to obtain an exact count, set this option to TRUE.	FALSE

Name	Description	Default
verbose_errors	When this option is set to TRUE, the PHP driver returns verbose errors. When this option is set to FALSE, you must call the sasql_error or sasql_errorcode functions to get further error information.	TRUE

You can change the default value for an option by including the following line in the *php.ini* file. In this example, the default value is set for the auto\_commit option.

```
sqlanywhere.auto_commit=0
```

**Parameters**

**\$conn** The connection resource returned by a connect function.

**\$option** The name of the option you want to set.

**\$value** The new option value.

**Returns**

TRUE on success or FALSE on failure.

**See also**

- [“sasql\\_commit”](#)
- [“sasql\\_error”](#)
- [“sasql\\_errorcode”](#)
- [“sasql\\_num\\_rows”](#)
- [“sasql\\_rollback”](#)

## sasql\_stmt\_affected\_rows

**Prototype**

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

**Description**

Returns the number of rows affected by executing the statement.

**Parameters**

**\$stmt** A statement resource that was executed by sasql\_stmt\_execute.

**Returns**

The number of rows affected or FALSE on failure.

**See also**

- [“sasql\\_stmt\\_execute”](#)

## `sasql_stmt_bind_param`

### Prototype

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types, mixed &$var_1 [, mixed &$var_2 .. ] )
```

### Description

Binds PHP variables to statement parameters.

### Parameters

**`$stmt`** A prepared statement resource that was returned by the `sasql_prepare` function.

**`$types`** A string that contains one or more characters specifying the types of the corresponding bind. This can be any of: **s** for string, **i** for integer, **d** for double, **b** for blobs. The length of the `$types` string must match the number of parameters that follow the `$types` parameter (`$var_1`, `$var_2`, ...). The number of characters should also match the number of parameter markers (question marks) in the prepared statement.

**`$var_n`** The variable references.

### Returns

TRUE if binding the variables was successful or FALSE otherwise.

### See also

- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_param\\_count”](#)
- [“sasql\\_stmt\\_bind\\_param\\_ex”](#)
- [“sasql\\_stmt\\_execute”](#)

## `sasql_stmt_bind_param_ex`

### Prototype

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int $param_number, mixed &$var, string $type [, bool $is_null [, int $direction ] ] )
```

### Description

Binds a PHP variable to a statement parameter.

### Parameters

**`$stmt`** A prepared statement resource that was returned by the `sasql_prepare` function.

**`$param_number`** The parameter number. This should be a number between 0 and (`sasql_stmt_param_count($stmt) - 1`).

**`$var`** A PHP variable. Only references to PHP variables are allowed.

**\$type** Type of the variable. This can be one of: **s** for string, **i** for integer, **d** for double, **b** for blobs.

**\$is\_null** Whether the value of the variable is NULL or not.

**\$direction** Can be SASQL\_D\_INPUT, SASQL\_D\_OUTPUT, or SASQL\_INPUT\_OUTPUT.

### Returns

TRUE if binding the variable was successful or FALSE otherwise.

### See also

- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_param\\_count”](#)
- [“sasql\\_stmt\\_bind\\_param”](#)
- [“sasql\\_stmt\\_execute”](#)

## sasql\_stmt\_bind\_result

### Prototype

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [, mixed &$var2 .. ] )
```

### Description

Binds one or more PHP variables to result columns of a statement that was executed, and returns a result set.

### Parameters

**\$stmt** A statement resource that was executed by `sasql_stmt_execute`.

**\$var1** References to PHP variables that will be bound to result set columns returned by the `sasql_stmt_fetch`.

### Returns

TRUE on success or FALSE on failure.

### See also

- [“sasql\\_stmt\\_execute”](#)
- [“sasql\\_stmt\\_fetch”](#)

## sasql\_stmt\_close

### Prototype

```
bool sasql_stmt_close( sasql_stmt $stmt )
```



**Description**

Closes the supplied statement resource and frees any resources associated with it. This function will also free any result objects that were returned by the `sasql_stmt_result_metadata`.

**Parameters**

**\$stmt** A prepared statement resource that was returned by the `sasql_prepare` function.

**Returns**

TRUE for success or FALSE on failure.

**See also**

- [“sasql\\_stmt\\_result\\_metadata”](#)
- [“sasql\\_prepare”](#)

## sasql\_stmt\_data\_seek

**Prototype**

```
bool sasql_stmt_data_seek( sasql_stmt $stmt, int $offset )
```

**Description**

This function seeks to the specified offset in the result set.

**Parameters**

**\$stmt** A statement resource.

**\$offset** The offset in the result set. This is a number between 0 and `(sasql_stmt_num_rows($stmt) - 1)`.

**Returns**

TRUE on success or FALSE failure.

**See also**

- [“sasql\\_stmt\\_num\\_rows”](#)

## sasql\_stmt\_errno

**Prototype**

```
int sasql_stmt_errno( sasql_stmt $stmt )
```

**Description**

Returns the error code for the most recently executed statement function using the specified statement resource.

### Parameters

**\$stmt** A prepared statement resource that was returned by the `sasql_prepare` function.

### Returns

An integer error code. For a list of error codes, see [“SQL Anywhere error messages sorted by SQLCODE”](#) [*Error Messages*].

### See also

- [“sasql\\_stmt\\_error”](#)
- [“sasql\\_error”](#)
- [“sasql\\_errorcode”](#)
- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_result\\_metadata”](#)

## sasql\_stmt\_error

### Prototype

```
string sasql_stmt_error( sasql_stmt $stmt )
```

### Description

Returns the error text for the most recently executed statement function using the specified statement resource.

### Parameters

**\$stmt** A prepared statement resource that was returned by the `sasql_prepare` function.

### Returns

A string describing the error. For a list of error messages, see [“SQL Anywhere error messages”](#) [*Error Messages*].

### See also

- [“sasql\\_stmt\\_erno”](#)
- [“sasql\\_error”](#)
- [“sasql\\_errorcode”](#)
- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_result\\_metadata”](#)

## sasql\_stmt\_execute

### Prototype

```
bool sasql_stmt_execute( sasql_stmt $stmt )
```

## Description

Executes the prepared statement. The `sasql_stmt_result_metadata` can be used to check whether the statement returns a result set.

## Parameters

**\$stmt** A prepared statement resource that was returned by the `sasql_prepare` function. Variables should be bound before calling `execute`.

## Returns

TRUE for success or FALSE on failure.

## See also

- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_param\\_count”](#)
- [“sasql\\_stmt\\_bind\\_param”](#)
- [“sasql\\_stmt\\_bind\\_param\\_ex”](#)
- [“sasql\\_stmt\\_result\\_metadata”](#)
- [“sasql\\_stmt\\_bind\\_result”](#)

# sasql\_stmt\_fetch

## Prototype

```
bool sasql_stmt_fetch( sasql_stmt $stmt )
```

## Description

This function fetches one row out of the result for the statement and places the columns in the variables that were bound using `sasql_stmt_bind_result`.

## Parameters

**\$stmt** A statement resource.

## Returns

TRUE on success or FALSE on failure.

## See also

- [“sasql\\_stmt\\_bind\\_result”](#)

# sasql\_stmt\_field\_count

## Prototype

```
int sasql_stmt_field_count( sasql_stmt $stmt )
```

### Description

This function returns the number of columns in the result set of the statement.

### Parameters

**\$stmt** A statement resource.

### Returns

The number of columns in the result of the statement. If the statement does not return a result, it returns 0.

### See also

- [“sasql\\_stmt\\_result\\_metadata”](#)

## sasql\_stmt\_free\_result

### Prototype

bool **sasql\_stmt\_free\_result**( sasql\_stmt *\$stmt* )

### Description

This function frees cached result set of the statement.

### Parameters

**\$stmt** A statement resource that was executed using `sasql_stmt_execute`.

### Returns

TRUE on success or FALSE on failure.

### See also

- [“sasql\\_stmt\\_execute”](#)
- [“sasql\\_stmt\\_store\\_result”](#)

## sasql\_stmt\_insert\_id

### Prototype

int **sasql\_stmt\_insert\_id**( sasql\_stmt *\$stmt* )

### Description

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

**Parameters**

**\$stmt** A statement resource that was executed by `sasql_stmt_execute`.

**Returns**

The ID generated for an IDENTITY column or a DEFAULT AUTOINCREMENT column by a previous INSERT statement, or zero if the last insert did not affect an IDENTITY or DEFAULT AUTOINCREMENT column. The function can return FALSE (0) if `$stmt` is not valid.

**See also**

- [“sasql\\_stmt\\_execute”](#)

## **sasql\_stmt\_next\_result**

**Prototype**

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

**Description**

This function advances to the next result from the statement. If there is another result set, the currently cached results are discarded and the associated result set object deleted (as returned by `sasql_stmt_result_metadata`).

**Parameters**

**\$stmt** A statement resource.

**Returns**

TRUE on success or FALSE failure.

**See also**

- [“sasql\\_stmt\\_result\\_metadata”](#)

## **sasql\_stmt\_num\_rows**

**Prototype**

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

**Description**

Returns the number of rows in the result set. The actual number of rows in the result set can only be determined after the `sasql_stmt_store_result` function is called to buffer the entire result set. If the `sasql_stmt_store_result` function has not been called, 0 is returned.

### Parameters

**\$stmt** A statement resource that was executed by `sasql_stmt_execute` and for which `sasql_stmt_store_result` was called.

### Returns

The number of rows available in the result or 0 on failure.

### See also

- [“sasql\\_stmt\\_execute”](#)
- [“sasql\\_stmt\\_store\\_result”](#)

## sasql\_stmt\_param\_count

### Prototype

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

### Description

Returns the number of parameters in the supplied prepared statement resource.

### Parameters

**\$stmt** A statement resource returned by the `sasql_prepare` function.

### Returns

The number of parameters or FALSE on error.

### See also

- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_bind\\_param”](#)
- [“sasql\\_stmt\\_bind\\_param\\_ex”](#)

## sasql\_stmt\_reset

### Prototype

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

### Description

This function resets the `$stmt` object to the state just after the describe. Any variables that were bound are unbound and any data sent using `sasql_stmt_send_long_data` are dropped.

### Parameters

**\$stmt** A statement resource.

**Returns**

TRUE on success or FALSE on failure.

**See also**

- [“sasql\\_stmt\\_send\\_long\\_data”](#)

## sasql\_stmt\_result\_metadata

**Prototype**

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

**Description**

Returns a result set object for the supplied statement.

**Parameters**

**\$stmt** A statement resource that was prepared and executed.

**Returns**

sasql\_result object or FALSE if the statement does not return any results.

## sasql\_stmt\_send\_long\_data

**Prototype**

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int $param_number, string $data )
```

**Description**

Allows the user to send parameter data in chunks. The user must first call `sasql_stmt_bind_param` or `sasql_stmt_bind_param_ex` before attempting to send any data. The bind parameter must be of type string or blob. Repeatedly calling this function appends on to what was previously sent.

**Parameters**

**\$stmt** A statement resource that was prepared using `sasql_prepare`.

**\$param\_number** The parameter number. This must be a number between 0 and `(sasql_stmt_param_count($stmt) - 1)`.

**\$data** The data to be sent.

**Returns**

TRUE on success or FALSE on failure.

### See also

- [“sasql\\_stmt\\_bind\\_param”](#)
- [“sasql\\_stmt\\_bind\\_param\\_ex”](#)
- [“sasql\\_prepare”](#)
- [“sasql\\_stmt\\_param\\_count”](#)

## sasql\_stmt\_store\_result

### Prototype

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

### Description

This function allows the client to cache the whole result set of the statement. You can use the function `sasql_stmt_free_result` to free the cached result.

### Parameters

**\$stmt** A statement resource that was executed using `sasql_stmt_execute`.

### Returns

TRUE on success or FALSE on failure.

### See also

- [“sasql\\_stmt\\_free\\_result”](#)
- [“sasql\\_stmt\\_execute”](#)

## sasql\_store\_result

### Prototype

```
sasql_result sasql_store_result( sasql_conn $conn )
```

### Description

Transfers the result set from the last query on the database connection `$conn` to be used with the `sasql_data_seek` function.

### Parameters

**\$conn** The connection resource returned by a connect function.

### Returns

FALSE if the query does not return a result object, or a result set object, that contains all the rows of the result. The result is cached at the client.



**See also**

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_stmt\\_execute”](#)

## sasql\_sqlstate

**Prototype**

```
string sasql_sqlstate( sasql_conn $conn )
```

**Description**

Returns the most recent SQLSTATE string. SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition. SQLSTATE codes consists of five characters with "00000" representing no error. The values are defined by the ISO/ANSI SQL standard.

**Parameters**

**\$conn** The connection resource returned by a connect function.

**Returns**

Returns a string of five characters containing the current SQLSTATE code. Note that "00000" means no error. For a list of SQLSTATE codes, see [“SQL Anywhere error messages sorted by SQLSTATE”](#) [*Error Messages*].

**See also**

- [“sasql\\_error”](#)
- [“sasql\\_errorcode”](#)

## sasql\_use\_result

**Prototype**

```
sasql_result sasql_use_result( sasql_conn $conn )
```

**Description**

Initiates a result set retrieval for the last query that executed on the connection.

**Parameters**

**\$conn** The connection resource returned by a connect function.

**Returns**

FALSE if the query does not return a result object or a result set object. The result is not cached on the client.

### See also

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_stmt\\_execute”](#)

## Deprecated PHP functions

The following PHP functions are supported but deprecated. Each of these functions has a newer equivalent with a name starting with `sasql_` instead of `sqlanywhere_`.

### `sqlanywhere_commit` (deprecated)

#### Prototype

```
bool sqlanywhere_commit( resource link_identifier )
```

#### Description

This function is deprecated. You should use the `sasql_commit` function instead. See [“sasql\\_commit” on page 643](#).

Ends a transaction on the SQL Anywhere database and makes any changes made during the transaction permanent. Useful only when the `auto_commit` option is Off.

#### Parameters

**link\_identifier** The link identifier returned by the `sqlanywhere_connect` function.

#### Returns

TRUE on success or FALSE on failure.

#### Example

This example shows how `sqlanywhere_commit` can be used to cause a commit on a specific connection.

```
$result = sqlanywhere_commit( $conn );
```

### See also

- [“sasql\\_commit”](#)
- [“sasql\\_pconnect”](#)
- [“sasql\\_disconnect”](#)
- [“sqlanywhere\\_pconnect \(deprecated\)”](#)
- [“sqlanywhere\\_disconnect \(deprecated\)”](#)

### `sqlanywhere_connect` (deprecated)

#### Prototype

```
resource sqlanywhere_connect( string con_str )
```

## Description

This function is deprecated. You should use the `sasql_connect` function instead. See [“sasql\\_connect” on page 643](#).

Establishes a connection to a SQL Anywhere database.

## Parameters

**con\_str** A connection string as recognized by SQL Anywhere.

## Returns

A positive SQL Anywhere link identifier on success, or an error and 0 on failure.

## Example

This example passes the user ID and password for a SQL Anywhere database in the connection string.

```
$conn = sqlanywhere_connect( "UID=DBA;PWD=sql" );
```

## See also

- [“sasql\\_connect”](#)
- [“sasql\\_pconnect”](#)
- [“sasql\\_disconnect”](#)
- [“sqlanywhere\\_pconnect \(deprecated\)”](#)
- [“sqlanywhere\\_disconnect \(deprecated\)”](#)

## sqlanywhere\_data\_seek (deprecated)

### Prototype

```
bool sqlanywhere_data_seek( resource result_identifier, int row_num )
```

### Description

This function is deprecated. You should use the `sasql_data_seek` function instead. See [“sasql\\_data\\_seek” on page 644](#).

Positions the cursor on row `row_num` on the `result_identifier` that was opened using `sqlanywhere_query`.

### Parameters

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

**row\_num** An integer that represents the new position of the cursor within the `result_identifier`. For example, specify 0 to move the cursor to the first row of the result set or 5 to move it to the sixth row. Negative numbers represent rows relative to the end of the result set. For example, -1 moves the cursor to the last row in the result set and -2 moves it to the second-last row.

### Returns

TRUE on success or FALSE on error.

## Example

This example shows how to seek to the sixth record in the result set.

```
sqlanywhere_data_seek( $result, 5 );
```

## See also

- [“sasql\\_data\\_seek”](#)
- [“sasql\\_fetch\\_field”](#)
- [“sasql\\_fetch\\_array”](#)
- [“sasql\\_fetch\\_row”](#)
- [“sasql\\_fetch\\_object”](#)
- [“sasql\\_query”](#)
- [“sqlanywhere\\_fetch\\_field \(deprecated\)”](#)
- [“sqlanywhere\\_fetch\\_array \(deprecated\)”](#)
- [“sqlanywhere\\_fetch\\_row \(deprecated\)”](#)
- [“sqlanywhere\\_fetch\\_object \(deprecated\)”](#)
- [“sqlanywhere\\_query \(deprecated\)”](#)

## sqlanywhere\_disconnect (deprecated)

### Prototype

```
bool sqlanywhere_disconnect( resource link_identifier )
```

### Description

This function is deprecated. You should use the [sasql\\_disconnect](#) instead. See [“sasql\\_disconnect” on page 645](#).

Closes a connection that has already been opened with [sqlanywhere\\_connect](#).

### Parameters

**link\_identifier** The link identifier returned by the [sqlanywhere\\_connect](#) function.

### Returns

TRUE on success or FALSE on error.

## Example

This example closes the connection to a database.

```
sqlanywhere_disconnect( $conn );
```

## See also

- [“sasql\\_disconnect”](#)
- [“sasql\\_connect”](#)
- [“sasql\\_pconnect”](#)
- [“sqlanywhere\\_connect \(deprecated\)”](#)
- [“sqlanywhere\\_pconnect \(deprecated\)”](#)

## sqlanywhere\_error (deprecated)

### Prototype

string `sqlanywhere_error`( [ resource *link\_identifier* ] )

### Description

This function is deprecated. You should use the `sasql_error` function instead. See [“sasql\\_error” on page 645](#).

Returns the error text of the most recently executed SQL Anywhere PHP function. Error messages are stored per connection. If no *link\_identifier* is specified, then `sqlanywhere_error` returns the last error message where no connection was available. For example, if you call `sqlanywhere_connect` and the connection fails, then call `sqlanywhere_error` with no parameter for *link\_identifier* to get the error message. If you want to obtain the corresponding SQL Anywhere error code value, use the `sqlanywhere_errorcode` function.

### Parameters

**link\_identifier** A link identifier that was returned by `sqlanywhere_connect` or `sqlanywhere_pconnect`.

### Returns

A string describing the error.

### Example

This example attempts to select from a table that does not exist. The `sqlanywhere_query` function returns FALSE and the `sqlanywhere_error` function returns the error message.

```
$result = sqlanywhere_query( $conn, "SELECT * FROM
table_that_does_not_exist" );
if( ! $result ) {
    $error_msg = sqlanywhere_error( $conn );
    echo "Query failed. Reason: $error_msg";
}
```

### See also

- [“sasql\\_error”](#)
- [“sasql\\_errorcode”](#)
- [“sasql\\_set\\_option”](#)
- [“sqlanywhere\\_errorcode \(deprecated\)”](#)
- [“sqlanywhere\\_set\\_option \(deprecated\)”](#)

## sqlanywhere\_errorcode (deprecated)

### Prototype

bool `sqlanywhere_errorcode`( [ resource *link\_identifier* ] )

## Description

This function is deprecated. You should use the `sasql_errorcode` function instead. See [“sasql\\_errorcode” on page 646](#).

Returns the error code of the most-recently executed SQL Anywhere PHP function. Error codes are stored per connection. If no *link\_identifier* is specified, then `sqlanywhere_errorcode` returns the last error code where no connection was available. For example, if you are calling `sqlanywhere_connect` and the connection fails, then call `sqlanywhere_errorcode` with no parameter for the *link\_identifier* to get the error code. If you want to get the corresponding error message use the `sqlanywhere_error` function.

## Parameters

**link\_identifier** A link identifier that was returned by `sqlanywhere_connect` or `sqlanywhere_pconnect`.

## Returns

An integer representing a SQL Anywhere error code. An error code of 0 means success. A positive error code indicates success with warnings. A negative error code indicates failure.

## Example

This example shows how you can retrieve the last error code from a failed SQL Anywhere PHP call.

```
$result = sqlanywhere_query( $conn, "SELECT * from
table_that_does_not_exist" );
if( ! $result ) {
    $error_code = sqlanywhere_errorcode( $conn );
    echo "Query failed: Error code: $error_code";
}
```

## See also

- [“sasql\\_error”](#)
- [“sasql\\_set\\_option”](#)
- [“sqlanywhere\\_error \(deprecated\)”](#)
- [“sqlanywhere\\_set\\_option \(deprecated\)”](#)

## sqlanywhere\_execute (deprecated)

### Prototype

```
bool sqlanywhere_execute( resource link_identifier, string sql_str )
```

### Description

This function is deprecated.

Prepares and executes the SQL query *sql\_str* on the connection identified by the *link\_identifier* that has already been opened using `sqlanywhere_connect` or `sqlanywhere_pconnect`. This function returns TRUE or FALSE depending on the outcome of the query execution. This function is suitable for queries that do not return result sets. If you are expecting a result set, use the `sqlanywhere_query` function instead.

## Parameters

**link\_identifier** A link identifier returned by `sqlanywhere_connect` or `sqlanywhere_pconnect`.

**sql\_str** A SQL statement supported by SQL Anywhere.

## Returns

TRUE if the query executed successfully, otherwise, FALSE and an error message.

## Example

This example shows how to execute a DDL statement using the `sqlanywhere_execute` function.

```
if( sqlanywhere_execute( $conn, "CREATE TABLE my_test_table( INT id )" ) ) {  
    // handle success  
} else {  
    // handle failure  
}
```

## See also

- [“sasql\\_query”](#)
- [“sqlanywhere\\_query \(deprecated\)”](#)

## sqlanywhere\_fetch\_array (deprecated)

### Prototype

```
array sqlanywhere_fetch_array( resource result_identifier )
```

### Description

This function is deprecated. You should use the `sasql_fetch_array` function instead. See [“sasql\\_fetch\\_array” on page 647](#).

Fetches one row from the result set. This row is returned as an array that can be indexed by the column names or by the column indexes.

### Parameters

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

### Returns

An array that represents a row from the result set, or FALSE when no rows are available.

### Example

This example shows how to retrieve all the rows in a result set. Each row is returned as an array.

```
$result = sqlanywhere_query( $conn, "SELECT GivenName, Surname FROM  
Employees" );  
While( ($row = sqlanywhere_fetch_array( $result )) ) {  
    echo " GivenName = " . $row["GivenName"] . " \n" ;  
        echo " Surname = $row[1] \n";  
}
```

**See also**

- “[sasql\\_fetch\\_array](#)”
- “[sasql\\_data\\_seek](#)”
- “[sasql\\_fetch\\_field](#)”
- “[sasql\\_fetch\\_row](#)”
- “[sasql\\_fetch\\_object](#)”
- “[sasql\\_query](#)”
- “[sqlanywhere\\_data\\_seek](#) (deprecated)”
- “[sqlanywhere\\_fetch\\_field](#) (deprecated)”
- “[sqlanywhere\\_fetch\\_row](#) (deprecated)”
- “[sqlanywhere\\_fetch\\_object](#) (deprecated)”
- “[sqlanywhere\\_query](#) (deprecated)”

## sqlanywhere\_fetch\_field (deprecated)

**Prototype**

object **sqlanywhere\_fetch\_field**( resource *result\_identifier* [, *field\_offset* ] )

**Description**

This function is deprecated. You should use the [sasql\\_fetch\\_field](#) function instead. See “[sasql\\_fetch\\_field](#)” on page 648.

Returns an object that contains information about a specific column.

**Parameters**

**result\_identifier** The result resource returned by the [sqlanywhere\\_query](#) function.

**field\_offset** An integer representing the column/field on which you want to retrieve information. Columns are zero based; to get the first column, specify the value 0. If this parameter is omitted, then the next field object is returned.

**Returns**

An object that has the following properties:

- **id** contains the field/column number.
- **name** contains the field/column name.
- **numeric** indicates whether the field is a numeric value.
- **length** returns field length.
- **type** returns field type.

**Example**

This example shows how to use [sqlanywhere\\_fetch\\_field](#) to retrieve all the column information for a result set.



```

$result = sqlanywhere_query($conn, "SELECT GivenName, Surname FROM
Employees");
while( ($field = sqlanywhere_fetch_field( $result )) ) {
    echo " Field ID = $field->id \n";
    echo " Field name = $field->name \n";
}

```

**See also**

- “sasql\_data\_seek”
- “sasql\_fetch\_field”
- “sasql\_fetch\_array”
- “sasql\_fetch\_row”
- “sasql\_fetch\_object”
- “sasql\_query”
- “sqlanywhere\_data\_seek (deprecated)”
- “sqlanywhere\_fetch\_array (deprecated)”
- “sqlanywhere\_fetch\_row (deprecated)”
- “sqlanywhere\_fetch\_object (deprecated)”
- “sqlanywhere\_query (deprecated)”

## sqlanywhere\_fetch\_object (deprecated)

**Prototype**

```
object sqlanywhere_fetch_object( resource result_identifier )
```

**Description**

This function is deprecated. You should use the `sasql_fetch_object` function instead. See [“sasql\\_fetch\\_object” on page 649](#).

Fetches one row from the result set. This row is returned as an object that can be indexed by the column name only.

**Parameters**

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

**Returns**

An object that represents a row from the result set, or FALSE when no rows are available.

**Example**

This example shows how to retrieve one row at a time from a result set as an object. Column names can be used as object members to access the column value.

```

$result = sqlanywhere_query( $conn, "SELECT GivenName, Surname FROM
Employees" );
While( ($row = sqlanywhere_fetch_object( $result )) ) {
    echo "$row->GivenName \n"; # output the data in the first column
only.
}

```

### See also

- “[sasql\\_data\\_seek](#)”
- “[sasql\\_fetch\\_field](#)”
- “[sasql\\_fetch\\_array](#)”
- “[sasql\\_fetch\\_row](#)”
- “[sasql\\_fetch\\_object](#)”
- “[sasql\\_query](#)”
- “[sqlanywhere\\_query \(deprecated\)](#)”
- “[sqlanywhere\\_data\\_seek \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_field \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_array \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_row \(deprecated\)](#)”

## sqlanywhere\_fetch\_row (deprecated)

### Prototype

```
array sqlanywhere_fetch_row( resource result_identifier )
```

### Description

This function is deprecated. You should use the [sasql\\_fetch\\_row](#) function instead. See “[sasql\\_fetch\\_row](#)” on page 649.

Fetches one row from the result set. This row is returned as an array that can be indexed by the column indexes only.

### Parameters

**result\_identifier** The result resource returned by the [sqlanywhere\\_query](#) function.

### Returns

An array that represents a row from the result set, or FALSE when no rows are available.

### Example

This example shows how to retrieve one row at a time from a result set.

```
while( ($row = sqlanywhere_fetch_row( $result )) ) {  
    echo "$row[0] \n";      # output the data in the first column only.  
}
```

**See also**

- “[sasql\\_fetch\\_row](#)”
- “[sasql\\_data\\_seek](#)”
- “[sasql\\_fetch\\_field](#)”
- “[sasql\\_fetch\\_array](#)”
- “[sasql\\_fetch\\_object](#)”
- “[sasql\\_query](#)”
- “[sqlanywhere\\_data\\_seek \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_field \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_array \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_object \(deprecated\)](#)”
- “[sqlanywhere\\_query \(deprecated\)](#)”

## sqlanywhere\_free\_result (deprecated)

**Prototype**

```
bool sqlanywhere_free_result( resource result_identifier )
```

**Description**

This function is deprecated. You should use the [sasql\\_free\\_result](#) function instead. See “[sasql\\_free\\_result](#)” on page 651.

Frees database resources associated with a result resource returned from [sqlanywhere\\_query](#).

**Parameters**

**result\_identifier** The result resource returned by the [sqlanywhere\\_query](#) function.

**Returns**

TRUE on success or FALSE on error.

**Example**

This example shows how to free a result identifier's resources.

```
sqlanywhere_free_result( $result );
```

**See also**

- “[sasql\\_query](#)”
- “[sasql\\_free\\_result](#)”
- “[sqlanywhere\\_query \(deprecated\)](#)”

## sqlanywhere\_identity (deprecated)

**Prototype**

```
int sqlanywhere_identity( resource link_identifier )
```

```
int sqlanywhere_insert_id( resource link_identifier )
```

### Description

This function is deprecated. You should use the `sasql_insert_id` function instead. See [“sasql\\_insert\\_id” on page 652](#).

Returns the last value inserted into an IDENTITY column or a DEFAULT AUTOINCREMENT column, or zero if the most recent insert was into a table that did not contain an IDENTITY or DEFAULT AUTOINCREMENT column.

The `sqlanywhere_insert_id` function is provided for compatibility with MySQL databases.

### Parameters

**link\_identifier** A link identifier returned by `sqlanywhere_connect` or `sqlanywhere_pconnect`.

### Returns

The ID generated for an AUTOINCREMENT column by a previous INSERT statement or zero if last insert did not affect an AUTOINCREMENT column. The function can return FALSE if the *link\_identifier* is not valid.

### Example

This example shows how the `sqlanywhere_identity` function can be used to retrieve the AUTOINCREMENT value most recently inserted into a table by the specified connection.

```
if( sqlanywhere_execute( $conn, "INSERT INTO my_auto_increment_table VALUES
( 1 ) " ) ) {
    $insert_id = sqlanywhere_insert_id( $conn );
    echo "Last insert id = $insert_id";
}
```

### See also

- [“sasql\\_insert\\_id”](#)
- [“sqlanywhere\\_execute \(deprecated\)”](#)

## sqlanywhere\_num\_fields (deprecated)

### Prototype

```
int sqlanywhere_num_fields( resource result_identifier )
```

### Description

This function is deprecated. You should use the `sasql_field_count` function instead. See [“sasql\\_field\\_count” on page 650](#).

Returns the number of columns (fields) the *result\_identifier* contains.

### Parameters

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

## Returns

A positive number of columns, or an error if *result\_identifier* is not valid.

## Example

This example returns a value indicating how many columns are in the result set.

```
$num_columns = sqlanywhere_num_fields( $result );
```

## See also

- [“sasql\\_field\\_count”](#)
- [“sasql\\_query”](#)
- [“sqlanywhere\\_query \(deprecated\)”](#)

## sqlanywhere\_num\_rows (deprecated)

### Prototype

```
int sqlanywhere_num_rows( resource result_identifier )
```

### Description

This function is deprecated. You should use the `sasql_num_rows` function instead. See [“sasql\\_num\\_rows” on page 654](#).

Returns the number of rows that the *result\_identifier* contains.

### Parameters

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

### Returns

A positive number if the number of rows is exact, or a negative number if it is an estimate. To get the exact number of rows, the database option `row_counts` must be set permanently on the database, or temporarily on the connection. See [“sasql\\_set\\_option” on page 659](#).

## Example

This example shows how to retrieve the estimated number of rows returned in a result set:

```
$num_rows = sqlanywhere_num_rows( $result );
    if( $num_rows < 0 ) {
        $num_rows = abs( $num_rows );    # take the absolute value as an
estimate
    }
```

## See also

- [“sasql\\_num\\_rows”](#)
- [“sasql\\_query”](#)
- [“sqlanywhere\\_query \(deprecated\)”](#)

## sqlanywhere\_pconnect (deprecated)

### Prototype

```
resource sqlanywhere_pconnect( string con_str )
```

### Description

This function is deprecated. You should use the `sasql_pconnect` function instead. See [“sasql\\_pconnect” on page 655](#).

Establishes a persistent connection to a SQL Anywhere database. Because of the way Apache creates child processes, you may observe a performance gain when using `sqlanywhere_pconnect` instead of `sqlanywhere_connect`. Persistent connections may provide improved performance in a similar fashion to connection pooling. If your database server has a limited number of connections (for example, the personal database server is limited to 10 concurrent connections), caution should be exercised when using persistent connections. Persistent connections could be attached to each of the child processes, and if you have more child processes in Apache than there are available connections, you will receive connection errors.

### Parameters

**con\_str** A connection string as recognized by SQL Anywhere.

### Returns

A positive SQL Anywhere persistent link identifier on success, or an error and 0 on failure.

### Example

This example shows how to retrieve all the rows in a result set. Each row is returned as an array.

```
$conn = sqlanywhere_pconnect( "UID=DBA;PWD=sql" );
```

### See also

- [“sasql\\_pconnect”](#)
- [“sasql\\_connect”](#)
- [“sasql\\_disconnect”](#)
- [“sqlanywhere\\_connect \(deprecated\)”](#)
- [“sqlanywhere\\_disconnect \(deprecated\)”](#)

## sqlanywhere\_query (deprecated)

### Prototype

```
resource sqlanywhere_query( resource link_identifier, string sql_str )
```

### Description

This function is deprecated. You should use the `sasql_query` function instead. See [“sasql\\_query” on page 656](#).

Prepares and executes the SQL query *sql\_str* on the connection identified by *link\_identifier* that has already been opened using `sqlanywhere_connect` or `sqlanywhere_pconnect`. For queries that do not return result sets, you can use the `sqlanywhere_execute` function.

### Parameters

**link\_identifier** The link identifier returned by the `sqlanywhere_connect` function.

**sql\_str** A SQL statement supported by SQL Anywhere.

For more information about SQL statements, see “SQL statements” [[SQL Anywhere Server - SQL Reference](#)].

### Returns

A positive value representing the result resource on success, or 0 and an error message on failure.

### Example

This example executes the query `SELECT * FROM SYSTAB` on the SQL Anywhere database.

```
$result = sqlanywhere_query( $conn, "SELECT * FROM SYSTAB" );
```

### See also

- “[sasql\\_query](#)”
- “[sasql\\_free\\_result](#)”
- “[sasql\\_fetch\\_array](#)”
- “[sasql\\_fetch\\_field](#)”
- “[sasql\\_fetch\\_object](#)”
- “[sasql\\_fetch\\_row](#)”
- “[sqlanywhere\\_execute \(deprecated\)](#)”
- “[sqlanywhere\\_free\\_result \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_array \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_field \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_object \(deprecated\)](#)”
- “[sqlanywhere\\_fetch\\_row \(deprecated\)](#)”

## sqlanywhere\_result\_all (deprecated)

### Prototype

```
bool sqlanywhere_result_all( resource result_identifier [, html_table_format_string [,  
html_table_header_format_string [, html_table_row_format_string [, html_table_cell_format_string ] ] ] ] )
```

### Description

This function is deprecated.

Fetches all results of the *result\_identifier* and generates an HTML output table with an optional formatting string.

## Parameters

**result\_identifier** The result resource returned by the `sqlanywhere_query` function.

**html\_table\_format\_string** A format string that applies to HTML tables. For example, "**Border=1; Cellpadding=5**". The special value `none` does not create an HTML table. This is useful if you want to customize your column names or scripts. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**html\_table\_header\_format\_string** A format string that applies to column headings for HTML tables. For example, "**bgcolor=#FF9533**". The special value `none` does not create an HTML table. This is useful if you want to customize your column names or scripts. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**html\_table\_row\_format\_string** A format string that applies to rows within HTML tables. For example, "**onclick='alert('this')'**". If you would like different formats that alternate, use the special token `><`. The left side of the token indicates which format to use on odd rows and the right side of the token is used to format even rows. If you do not place this token in your format string, all rows have the same format. If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

**html\_table\_cell\_format\_string** A format string that applies to cells within HTML table rows. For example, "**onclick='alert('this')'**". If you do not want to specify an explicit value for this parameter, use `NULL` for the parameter value.

## Returns

`TRUE` on success or `FALSE` on failure.

## Example

This example shows how to use `sqlanywhere_result_all` to generate an HTML table with all the rows from a result set.

```
$result = sqlanywhere_query( $conn, "SELECT GivenName, Surname FROM Employees" );
sqlanywhere_result_all( $result );
```

This example shows how to use different formatting on alternate rows using a style sheet.

```
$result = sqlanywhere_query( $conn, "SELECT GivenName, Surname FROM Employees" );
sqlanywhere_result_all( $result, "border=2", "bordercolor=#3F3986",
"bgcolor=#3F3986 style=\"color=#FF9533\"", 'class="even"><class="odd"');
```

## See also

- [“sasql\\_query”](#)
- [“sqlanywhere\\_query \(deprecated\)”](#)



## sqlanywhere\_rollback (deprecated)

### Prototype

```
bool sqlanywhere_rollback( resource link_identifier )
```

### Description

This function is deprecated. You should use the `sasql_rollback` function instead. See [“sasql\\_rollback” on page 659](#).

Ends a transaction on the SQL Anywhere database and discards any changes made during the transaction. This function is only useful when the `auto_commit` option is Off.

### Parameters

**link\_identifier** The link identifier returned by the `sqlanywhere_connect` function.

### Returns

TRUE on success or FALSE on failure.

### Example

This example uses `sqlanywhere_rollback` to roll back a connection.

```
$result = sqlanywhere_rollback( $conn );
```

### See also

- [“sasql\\_rollback”](#)
- [“sasql\\_commit”](#)
- [“sasql\\_set\\_option”](#)
- [“sqlanywhere\\_commit \(deprecated\)”](#)
- [“sqlanywhere\\_set\\_option \(deprecated\)”](#)

## sqlanywhere\_set\_option (deprecated)

### Prototype

```
bool sqlanywhere_set_option( resource link_identifier, string option, mixed value )
```

### Description

This function is deprecated. You should use the `sasql_set_option` function instead. See [“sasql\\_set\\_option” on page 659](#).

Sets the value of the specified option on the specified connection. You can set the value for the following options:

Name	Description	Default
auto_commit	When this option is set to on, the database server commits after executing each statement.	on
row_counts	When this option is set to FALSE, the <code>sqlanywhere_num_rows</code> function returns an estimate of the number of rows affected. If you want to obtain an exact count, set this option to TRUE.	FALSE
verbose_errors	When this option is set to TRUE, the PHP driver returns verbose errors. When this option is set to FALSE, you must call the <code>sqlanywhere_error</code> or <code>sqlanywhere_errorcode</code> functions to get further error information.	TRUE

You can change the default value for an option by including the following line in the `php.ini` file. In this example, the default value is set for the `auto_commit` option.

```
sqlanywhere.auto_commit=0
```

### Parameters

**link\_identifier** The link identifier returned by the `sqlanywhere_connect` function.

**option** The name of the option you want to set.

**value** The new option value.

### Returns

TRUE on success or FALSE on failure.

### Example

The following examples show the different ways you can set the value of the `auto_commit` option.

```
$result = sqlanywhere_set_option( $conn, "auto_commit", "Off" );  
$result = sqlanywhere_set_option( $conn, "auto_commit", 0 );  
$result = sqlanywhere_set_option( $conn, "auto_commit", False );
```

**See also**

- `“sasql_set_option”`
- `“sasql_commit”`
- `“sasql_error”`
- `“sasql_errorcode”`
- `“sasql_num_rows”`
- `“sasql_rollback”`
- `“sqlanywhere_commit (deprecated)”`
- `“sqlanywhere_error (deprecated)”`
- `“sqlanywhere_errorcode (deprecated)”`
- `“sqlanywhere_num_rows (deprecated)”`
- `“sqlanywhere_rollback (deprecated)”`

---

---

# Ruby support

## SQL Anywhere Ruby API support

There are three different Ruby APIs supported by SQL Anywhere. First, there is the SQL Anywhere Ruby API. This API provides a Ruby wrapping over the interface exposed by the SQL Anywhere C API. Second, there is support for ActiveRecord, an object-relational mapper popularized by being part of the Ruby on Rails web development framework. Third, there is support for Ruby DBI. SQL Anywhere provides a Ruby Database Driver (DBD) which can be used with DBI.

There are three separate packages available in the SQL Anywhere for Ruby project. The simplest way to install any of these packages is to use **RubyGems**. To obtain RubyGems, go to <http://rubyforge.org/projects/rubygems/>.

The home for the SQL Anywhere Ruby project is <http://sqlanywhere.rubyforge.org/>.

### SQL Anywhere Native Ruby Driver

**sqlanywhere** This package is a low-level driver that allows Ruby code to interface with SQL Anywhere databases. This package provides a Ruby wrapping over the interface exposed by the SQL Anywhere C API. This package is written in C and is available as source, or as pre-compiled gems, for Windows and Linux. If you have **RubyGems** installed, this package can be obtained by running the following command:

```
gem install sqlanywhere
```

This package is a prerequisite for any of the other SQL Anywhere Ruby packages.

### See also

- “SQL Anywhere Ruby API reference” on page 699
- Download source: <http://rubyforge.org/projects/sqlanywhere>
- RDocs: <http://sqlanywhere.rubyforge.org/sqlanywhere/>
- Ruby Programming Language: <http://www.ruby-lang.org>
- RubyForge/Ruby Central: <http://rubyforge.org/>

### SQL Anywhere ActiveRecord Adapter

**activerecord-sqlanywhere-adapter** This package is an adapter that allows ActiveRecord to communicate with SQL Anywhere. ActiveRecord is an object-relational mapper, popularized by being part of the Ruby on Rails web development framework. This package is written in pure Ruby, and available in source, or gem format. This adapter uses (and has a dependency on) the **sqlanywhere** gem. If you have **RubyGems** installed, this package and its dependencies can be installed by running the following command:

```
gem install activerecord-sqlanywhere-adapter
```

### See also

- “Rails support in SQL Anywhere”
- Download source: <http://rubyforge.org/projects/sqlanywhere>
- RDocs: <http://sqlanywhere.rubyforge.org/activerecord-sqlanywhere-adapter>
- Ruby on Rails: <http://www.rubyonrails.org/>

### SQL Anywhere Ruby/DBI Driver

**dbi** This package is a DBI driver for Ruby. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbi
```

**dbd-sqlanywhere** This package is a driver that allows Ruby/DBI to communicate with SQL Anywhere. Ruby/DBI is a generic database interface modeled after Perl's popular DBI module. This package is written in pure Ruby, and available in source, or gem format. This driver uses (and has a dependency on) the sqlanywhere gem. If you have RubyGems installed, this package and its dependencies can be installed by running the following command:

```
gem install dbd-sqlanywhere
```

For feedback on any of these packages, use the mailing list [sqlanywhere-users@rubyforge.com](mailto:sqlanywhere-users@rubyforge.com).

For general questions about using SQL Anywhere in a web environment, use the SQL Anywhere Web Development at <http://groups.google.com/group/sql-anywhere-web-development> forum.

For general questions on SQL Anywhere and its usage, use the SQL Anywhere forum <http://sqlanywhere-forum.sybase.com/>.

### See also

- “Ruby-DBI Driver for SQL Anywhere”
- Download source: <http://rubyforge.org/projects/sqlanywhere>
- RDocs: <http://sqlanywhere.rubyforge.org/dbd-sqlanywhere>
- Ruby/DBI - Direct database access layer for Ruby: <http://ruby-dbi.rubyforge.org/>

## Rails support in SQL Anywhere

Rails is a web development framework written in the Ruby language. Its strength is in web application development. A familiarity with the Ruby programming language is highly recommended before you attempt Rails development. As part of becoming familiar with Ruby, you might consider reading “[SQL Anywhere Ruby API reference](#)” on page 699.

### Prerequisites for Rails development

- **RubyGems** You should install RubyGems. It simplifies the installation of Ruby packages. The Ruby on Rails download page directs you to the correct version to install. See <http://www.rubyonrails.org/>.

- **Ruby** You must install the Ruby interpreter on your system. The Ruby on Rails download page recommends which version to install. See <http://www.rubyonrails.org/>.
- **Rails** With RubyGems, you can install all of Rails and its dependencies with a single command line:

```
gem install rails
```

- **Ruby Development Kit** Install the Ruby Development Kit (**DevKit**). Download the **DevKit** from <http://rubyinstaller.org/downloads/> and follow the instructions at <http://github.com/oneclick/rubyinstaller/wiki/Development-Kit>.
- **activerecord-sqlanywhere-adapter** If you have not already done so, you must install the SQL Anywhere ActiveRecord support to do Rails development using SQL Anywhere. With RubyGems, you can install all of SQL Anywhere ActiveRecord support and its dependencies with a single command line:

```
gem install activerecord-sqlanywhere-adapter
```

## Before you begin

Once you have installed the requisite components, there are a few final steps that you must undertake before you can begin Rails development using SQL Anywhere. These steps are required to add SQL Anywhere to the set of database management systems supported by Rails. At the time of writing, Rails 3.1.3 was the current released version.

## Add SQL Anywhere support to Rails

1. To configure a database, you must create a *sqlanywhere.yml* file in the Rails *configs/databases* directory. If you have installed Ruby in the path *\Ruby* and you have installed version 3.1.3 of Rails, then the path to this file would be *\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\rails\app\templates\config\databases*. The contents of this file should be:

```
#
# SQL Anywhere database configuration
#
# This configuration file defines the patten used for
# database filenames. If your application is called "blog",
# then the database names will be blog_development,
# blog_test, blog_production. The specified username and
# password should permit DBA access to the database.
#
development:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_development
  username: DBA
  password: sql

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_test
```

```
username: DBA
password: sql

production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: sql
```

2. You must update the Rails *app\_base.rb* file. Using the same assumptions in step 1 above, this file is located in the path `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\app_base.rb`. Edit the *app\_base.rb* file and locate the following line:

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase ibm_db
sqlserver )
```

Add **sqlanywhere** to the list as follows.

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3 frontbase
ibm_db sqlserver )
```

Now save the file and exit.

The *sqlanywhere.yml* provides a template for creating *database.yml* files in Rails projects. The template above shows some of the database options that may be specified. Here is a complete list.

- **adapter** (required, no default). This option must be set to **sqlanywhere** to use the SQL Anywhere ActiveRecord adapter.
- **database** (required, no default). This option corresponds to "DatabaseName=" in a connection string.
- **server** (optional, defaults to the **database** option). This option corresponds to "ServerName=" in a connection string.
- **username** (optional, defaults to 'DBA'). This option corresponds to "UserID=" in a connection string.
- **password** (optional, defaults to 'sql'). This option corresponds to "Password=" in a connection string.
- **encoding** (optional, defaults to the OS character set). This option corresponds to "CharSet=" in a connection string.
- **commlinks** (optional). This option corresponds to "CommLinks=" in a connection string.
- **connection\_name** (optional). This option corresponds to "ConnectionName=" in connection string.

## Learning Rails

It is recommended that you start with the tutorial on the Ruby on Rails website. Use the notes below in parallel with the tutorial. These notes outline any differences when using SQL Anywhere. For the Rails tutorial, see [http://guides.rails.info/getting\\_started.html](http://guides.rails.info/getting_started.html).



In the tutorial, you are shown the command to initialize the **blog** project. Here is the command to initialize the **blog** project for use with SQL Anywhere:

```
rails new blog -d sqlanywhere
```

After you create the *blog* application, switch to its folder to continue work directly in that application:

```
cd blog
```

Edit the *gemfile* to include a **gem** directive for the SQL Anywhere ActiveRecord adapter. Add the new directive following the indicated line below:

```
gem 'sqlanywhere'  
gem 'activerecord-sqlanywhere-adapter'
```

If you look at the contents of the *config\database.yml* file, you will see that it references development, test, and production databases. Instead of using a **rake** command to create the databases as indicated by the tutorial, change to the *db* directory of the project and create three databases as follows.

```
cd db  
dbinit blog_development  
dbinit blog_test  
dbinit blog_production  
cd ..
```

Before you continue, you must start the database server and the three databases as follows.

```
dsrv12 -n blog blog_development.db blog_production.db blog_test.db
```

The server name in the command line (*blog*) must match the name specified by the **server:** tags in the *database.yml* file. The *sqlanywhere.yml* template file is configured to ensure that the server name matches the project name in all generated *database.yml* files.

### See also

- <http://www.rubyonrails.org/>

## Ruby-DBI Driver for SQL Anywhere

This section provides an overview of how to write Ruby applications that use the SQL Anywhere DBI driver. Complete documentation for the DBI module is available online at <http://ruby-dbi.rubyforge.org/>.

### Loading the DBI module

To use the DBI:SQLAnywhere interface from a Ruby application, you must first tell Ruby that you plan to use the Ruby DBI module. To do so, include the following line near the top of the Ruby source file.

```
require 'dbi'
```

The DBI module automatically loads the SQL Anywhere database driver (DBD) interface as required.

### Opening and closing a connection

Generally, you open a single connection to a database and then perform all the required operations through it by executing a sequence of SQL statements. To open a connection, you use the connect

function. The return value is a handle to the database connection that you use to perform subsequent operations on that connection.

The call to the connect function takes the general form:

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password, options)
```

- **server-name** is the name of the database server that you want to connect to. Alternately, you can specify a connection string in the format "option1=value1;option2=value2;...".
- **user-id** is a valid user ID. Unless this string is empty, ";UID=value" is appended to the connection string.
- **password** is the corresponding password for the user ID. Unless this string is empty, ";PWD=value" is appended to the connection string.
- **options** is a hash of additional connection parameters such as DatabaseName, DatabaseFile, and ConnectionName. These are appended to the connection string in the format "option1=value1;option2=value2;...".

To demonstrate the connect function, start the database server and sample database before running the sample Ruby scripts.

```
dbeng12 "%SQLANYSAMPl2%\demo.db"
```

The following code sample opens and closes a connection to the SQL Anywhere sample database. The string "demo" in the example below is the server name.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

Optionally, you can specify a connection string in place of the server name. For example, in the above script may be altered by replacing the first parameter to the connect function as follows:

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:SERVER=demo;DBN=demo', 'DBA', 'sql') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
    dbh.disconnect()
  end
end
```

The user ID and password cannot be specified in the connection string. Ruby DBI will automatically fill in the username and password with defaults if these arguments are omitted, so you should never include a UID or PWD connection parameter in your connection string. If you do, an exception will be thrown.

The following example shows how additional connection parameters can be passed to the connect function as a hash of key/value pairs.

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql',
```

```

        { :ConnectionName => "RubyDemo",
          :DatabaseFile => "demo.db",
          :DatabaseName => "demo" }
      ) do |dbh|
    if dbh.ping
      print "Successfully Connected\n"
      dbh.disconnect()
    end
  end
end

```

## Selecting data

Once you have obtained a handle to an open connection, you can access and modify data stored in the database. Perhaps the simplest operation is to retrieve some rows and print them out.

A SQL statement must be executed first. If the statement returns a result set, you use the resulting statement handle to retrieve meta information about the result set and the rows of the result set. The following example obtains the column names from the metadata and displays the column names and values for each row fetched.

```

require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields:  #{sth.column_names.size}\n"
  sth.fetch do |row|
    print "\n"
    sth.column_info.each_with_index do |info, i|
      unless info["type_name"] == "LONG BINARY"
        print "#{info["name"]}={row[i]}\n"
      end
    end
  end
  sth.finish
end

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql')
  db_query(dbh, "SELECT * FROM Products")
rescue DBI::DatabaseError => e
  puts "An error occurred"
  puts "Error code:  #{e.err}"
  puts "Error message:  #{e.errstr}"
  puts "Error SQLSTATE:  #{e.state}"
ensure
  dbh.disconnect if dbh
end

```

The first few lines of output that appear are reproduced below.

```

# of Fields:  8

ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301

```

```
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

It is important to call `finish` to release the statement handle when you are done. If you do not, then you may get an error like the following:

```
Resource governor for 'prepared statements' exceeded
```

To detect handle leaks, the SQL Anywhere database server limits the number of cursors and prepared statements permitted to a maximum of 50 per connection by default. The resource governor automatically generates an error if these limits are exceeded. If you get this error, check for undestroyed statement handles. Use `prepare_cached` sparingly, as the statement handles are not destroyed.

If necessary, you can alter these limits by setting the `max_cursor_count` and `max_statement_count` options.

### Inserting rows

Inserting rows requires a handle to an open connection. The simplest way to insert rows is to use a parameterized `INSERT` statement, meaning that question marks are used as placeholders for values. The statement is first prepared, and then executed once per new row. The new row values are supplied as parameters to the `execute` method.

```
require 'dbi'

def db_query( dbh, sql )
  sth = dbh.execute(sql)
  print "# of Fields:  #{sth.column_names.size}\n"
  sth.fetch do |row|
    print "\n"
    sth.column_info.each_with_index do |info, i|
      unless info["type_name"] == "LONG VARBINARY"
        print "#{info["name"]}={row[i]}\n"
      end
    end
  end
  sth.finish
end

def db_insert( dbh, rows )
  sql = "INSERT INTO Customers (ID, GivenName, Surname,
    Street, City, State, Country, PostalCode,
    Phone, CompanyName)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
  sth = dbh.prepare(sql);
  rows.each do |row|
    sth.execute(row[0],row[1],row[2],row[3],row[4],
      row[5],row[6],row[7],row[8],row[9])
  end
end

begin
  dbh = DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql')
  rows = [
    [801,'Alex','Alt','5 Blue Ave','New York','NY','USA',
      '10012','5185553434','BXM'],
```

```

        [802, 'Zach', 'Zed', '82 Fair St', 'New York', 'NY', 'USA',
          '10033', '5185552234', 'Zap']
    ]
    db_insert(dbh, rows)
    dbh.commit
    db_query(dbh, "SELECT * FROM Customers WHERE ID > 800")
  rescue DBI::DatabaseError => e
    puts "An error occurred"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
    puts "Error SQLSTATE: #{e.state}"
  ensure
    dbh.disconnect if dbh
  end
end

```

**See also**

- “max\_cursor\_count option” [[SQL Anywhere Server - Database Administration](#)]
- “max\_statement\_count option” [[SQL Anywhere Server - Database Administration](#)]

## SQL Anywhere Ruby API reference

SQL Anywhere provides a low-level interface to the SQL Anywhere C API. The API described in the following sections permits the rapid development of SQL applications. To demonstrate the power of Ruby application development, consider the following sample Ruby program. It loads the SQL Anywhere Ruby extension, connects to the sample database, lists column values from the Products table, disconnects, and terminates.

```

begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end

api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
conn = api.sqlany_new_connection()
api.sqlany_connect( conn, "DSN=SQL Anywhere 12 Demo" )
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.sqlany_num_rows( stmt )
num_rows.times {
  api.sqlany_fetch_next( stmt )
  num_cols = api.sqlany_num_cols( stmt )
  for col in 1..num_cols do
    info = api.sqlany_get_column_info( stmt, col - 1 )
    unless info[3]==1 # Don't do binary
      rc, value = api.sqlany_get_column( stmt, col - 1 )
      print "#{info[2]}=#{value}\n"
    end
  end
  print "\n"
}
api.sqlany_free_stmt( stmt )
api.sqlany_disconnect( conn )
api.sqlany_free_connection( conn )

```

```
api.sqlany_fini()  
SQLAnywhere::API.sqlany_finalize_interface( api )
```

The first two rows of the result set output from this Ruby program are shown below:

```
ID=300  
Name=Tee Shirt  
Description=Tank Top  
Size=Small  
Color=White  
Quantity=28  
UnitPrice=9.00
```

```
ID=301  
Name=Tee Shirt  
Description=V-neck  
Size=Medium  
Color=Orange  
Quantity=54  
UnitPrice=14.00
```

The following sections describe each of the supported functions.

## sqlany\_affected\_rows

Returns the number of rows affected by execution of the prepared statement.

### Syntax

```
sqlany_affected_rows ( $stmt )
```

### Parameters

- **\$stmt** A statement that was prepared and executed successfully in which no result set was returned. For example, an INSERT, UPDATE or DELETE statement was executed.

### Returns

Returns a scalar value that is the number of rows affected, or -1 on failure.

### See also

- [“sqlany\\_execute function” on page 705](#)

### Example

```
affected = api.sqlany_affected( stmt )
```

## sqlany\_bind\_param function

Binds a user-supplied buffer as a parameter to the prepared statement.

### Syntax

```
sqlany_bind_param ( $stmt, $index, $param )
```

**Parameters**

- **\$stmt** A statement object returned by the successful execution of `sqlany_prepare`.
- **\$index** The index of the parameter. The number must be between 0 and `sqlany_num_params() - 1`.
- **\$param** A filled bind object retrieved from `sqlany_describe_bind_param`.

**Returns**

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

**See also**

- [“sqlany\\_describe\\_bind\\_param function” on page 703](#)

**Example**

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
    SET Contacts.ID = Contacts.ID + 1000
    WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

## sqlany\_clear\_error function

Clears the last stored error code.

**Syntax**

```
sqlany_clear_error ( $conn )
```

**Parameters**

- **\$conn** A connection object returned from `sqlany_new_connection`.

**Returns**

Returns nil.

**See also**

- [“sqlany\\_new\\_connection function” on page 713](#)

**Example**

```
api.sqlany_clear_error( conn )
```

## sqlany\_client\_version function

Returns the current client version.

### Syntax

```
sqlany_client_version ( )
```

### Returns

Returns a scalar value that is the client version string.

### Example

```
buffer = api.sqlany_client_version()
```

## sqlany\_commit function

Commits the current transaction.

### Syntax

```
sqlany_commit ( $conn )
```

### Parameters

- **\$conn** The connection object on which the commit operation is to be performed.

### Returns

Returns a scalar value that is 1 when successful or 0 when unsuccessful.

### See also

- [“sqlany\\_rollback function” on page 716](#)

### Example

```
rc = api.sqlany_commit( conn )
```

## sqlany\_connect function

Creates a connection to a SQL Anywhere database server using the specified connection object and connection string.

### Syntax

```
sqlany_connect ( $conn, $str )
```

### Parameters

- **\$conn** The connection object created by `sqlany_new_connection`.
- **\$str** A SQL Anywhere connection string.



## Returns

Returns a scalar value that is 1 if the connection is established successfully or 0 when the connection fails. Use `sqlany_error` to retrieve the error code and message.

## See also

- [“sqlany\\_new\\_connection function” on page 713](#)
- [“sqlany\\_error function” on page 704](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“SQL Anywhere database connections” \[SQL Anywhere Server - Database Administration\]](#)

## Example

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Connection status = #{status}\n"
```

# sqlany\_describe\_bind\_param function

Describes the bind parameters of a prepared statement.

## Syntax

```
sqlany_describe_bind_param ( $stmt, $index )
```

## Parameters

- **\$stmt** A statement prepared successfully using `sqlany_prepare`.
- **\$index** The index of the parameter. The number must be between 0 and `sqlany_num_params()` - 1.

## Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

## Remarks

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement (stored procedure or a DML), determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

## See also

- [“sqlany\\_bind\\_param function” on page 700](#)
- [“sqlany\\_prepare function” on page 716](#)

## Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000"
```

```
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

## sqlany\_disconnect function

Disconnects a SQL Anywhere connection. All uncommitted transactions are rolled back.

### Syntax

```
sqlany_disconnect ( $conn )
```

### Parameters

- **\$conn** A connection object with a connection established using `sqlany_connect`.

### Returns

Returns a scalar value that is 1 on success or 0 on failure.

### See also

- [“sqlany\\_connect function” on page 702](#)
- [“sqlany\\_new\\_connection function” on page 713](#)

### Example

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

## sqlany\_error function

Returns the last error code and message stored in the connection object.

### Syntax

```
sqlany_error ( $conn )
```

### Parameters

- **\$conn** A connection object returned from `sqlany_new_connection`.

### Returns

Returns a 2-element array that contains the SQL error code as the first element and an error message string as the second element.

For the error code, positive values are warnings, negative values are errors, and 0 is success.

**See also**

- [“sqlany\\_connect function” on page 702](#)
- [“SQL Anywhere error messages sorted by SQLCODE” \[Error Messages\]](#)

**Example**

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

## sqlany\_execute function

Executes a prepared statement.

**Syntax**

```
sqlany_execute ( $stmt )
```

**Parameters**

- **\$stmt** A statement prepared successfully using `sqlany_prepare`.

**Returns**

Returns a scalar value that is 1 on success or 0 on failure.

**Remarks**

You can use `sqlany_num_cols` to verify if the statement returned a result set.

**See also**

- [“sqlany\\_prepare function” on page 716](#)

**Example**

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

## sqlany\_execute\_direct function

Executes the SQL statement specified by the string argument.

**Syntax**

```
sqlany_execute_direct ( $conn, $sql )
```

**Parameters**

- **\$conn** A connection object with a connection established using `sqlany_connect`.

- **\$sql** A SQL string. The SQL string should not have parameters such as ?.

### Returns

Returns a statement object or nil on failure.

### Remarks

Use this function if you want to prepare and execute a statement in one step. Do not use this function to execute a SQL statement with parameters.

### See also

- [“sqlany\\_fetch\\_absolute function” on page 707](#)
- [“sqlany\\_fetch\\_next function” on page 707](#)
- [“sqlany\\_num\\_cols function” on page 714](#)
- [“sqlany\\_get\\_column function” on page 710](#)

### Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ", ", managerID, ", ",
      surname, ", ", givenName, ", ", departmentID, "\n"
```

## sqlany\_execute\_immediate function

Executes the specified SQL statement immediately without returning a result set. It is useful for statements that do not return result sets.

### Syntax

```
sqlany_execute_immediate ( $conn, $sql )
```

### Parameters

- **\$conn** A connection object with a connection established using `sqlany_connect`.
- **\$sql** A SQL string. The SQL string should not have parameters such as ?.

### Returns

Returns a scalar value that is 1 on success or 0 on failure.

### See also

- [“sqlany\\_error function” on page 704](#)

**Example**

```
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= 50" )
```

## sqlany\_fetch\_absolute function

Moves the current row in the result set to the row number specified and then fetches the data at that row.

**Syntax**

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

**Parameters**

- **\$stmt** A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.
- **\$row\_num** The row number to be fetched. The first row is 1, the last row is -1.

**Returns**

Returns a scalar value that is 1 on success or 0 on failure.

**See also**

- [“sqlany\\_error function” on page 704](#)
- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)
- [“sqlany\\_fetch\\_next function” on page 707](#)

**Example**

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_absolute( stmt, 2 )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

## sqlany\_fetch\_next function

Returns the next row from the result set. This function first advances the row pointer and then fetches the data at the new row.

**Syntax**

```
sqlany_fetch_next ( $stmt )
```

## Parameters

- **\$stmt** A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.

## Returns

Returns a scalar value that is 1 on success or 0 on failure.

## See also

- [“sqlany\\_error function” on page 704](#)
- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)
- [“sqlany\\_fetch\\_absolute function” on page 707](#)

## Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
      surname, ",", givenName, ",", departmentID, "\n"
```

## sqlany\_fini function

Frees resources allocated by the API.

## Syntax

```
sqlany_fini ( )
```

## Returns

Returns nil.

## See also

- [“sqlany\\_init function” on page 713](#)

## Example

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

## sqlany\_free\_connection function

Frees the resources associated with a connection object.

### Syntax

```
sqlany_free_connection ( $conn )
```

### Parameters

- **\$conn** A connection object created by `sqlany_new_connection`.

### Returns

Returns nil.

### See also

- [“sqlany\\_new\\_connection function” on page 713](#)

### Example

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

## sqlany\_free\_stmt function

Frees resources associated with a statement object.

### Syntax

```
sqlany_free_stmt ( $stmt )
```

### Parameters

- **\$stmt** A statement object returned by the successful execution of `sqlany_prepare` or `sqlany_execute_direct`.

### Returns

Returns nil.

### See also

- [“sqlany\\_prepare function” on page 716](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)

## Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
rc = api.sqlany_free_stmt( stmt )
```

## sqlany\_get\_bind\_param\_info function

Retrieves information about the parameters that were bound using `sqlany_bind_param`.

### Syntax

```
sqlany_get_bind_param_info ( $stmt, $index )
```

### Parameters

- **\$stmt** A statement successfully prepared using `sqlany_prepare`.
- **\$index** The index of the parameter. The number must be between 0 and `sqlany_num_params()` - 1.

### Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and a described parameter as the second element.

### See also

- [“sqlany\\_bind\\_param function” on page 700](#)
- [“sqlany\\_describe\\_bind\\_param function” on page 703](#)
- [“sqlany\\_prepare function” on page 716](#)

## Example

```
# Get information on first parameter (0)
rc, param_info = api.sqlany_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "\n"
print "Param_info output = ", param_info.get_output(), "\n"
```

## sqlany\_get\_column function

Returns the value fetched for the specified column.

### Syntax

```
sqlany_get_column ( $stmt, $col_index )
```

### Parameters

- **\$stmt** A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.



- **\$col\_index** The number of the column to be retrieved. A column number is between 0 and `sqlany_num_cols() - 1`.

### Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the column value as the second element.

### See also

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)
- [“sqlany\\_fetch\\_absolute function” on page 707](#)
- [“sqlany\\_fetch\\_next function” on page 707](#)

### Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ", ", managerID, ", ",
      surname, ", ", givenName, ", ", departmentID, "\n"
```

## sqlany\_get\_column\_info function

Gets column information for the specified result set column.

### Syntax

```
sqlany_get_column_info ( $stmt, $col_index )
```

### Parameters

- **\$stmt** A statement object that was executed by `sqlany_execute` or `sqlany_execute_direct`.
- **\$col\_index** The column number between 0 and `sqlany_num_cols() - 1`.

### Returns

Returns a 9-element array of information describing a column in a result set. The first element contains 1 on success or 0 on failure. The array elements are described in the following table.

Element number	Type	Description
0	Integer	1 on success or 0 on failure.
1	Integer	Column index (0 to <code>sqlany_num_cols() - 1</code> ).

Element number	Type	Description
2	String	Column name.
3	Integer	Column type. See <a href="#">“Column types” on page 717</a> .
4	Integer	Column native type. See <a href="#">“Native column types” on page 718</a> .
5	Integer	Column precision (for numeric types).
6	Integer	Column scale (for numeric types).
7	Integer	Column size.
8	Integer	Column nullable (1=nullable, 0=not nullable).

**See also**

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)
- [“sqlany\\_prepare function” on page 716](#)

**Example**

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision, col_scale,
  col_size, col_nullable = api.sqlany_get_column_info( stmt, 0 )
```

## sqlany\_get\_next\_result function

Advances to the next result set in a multiple result set query.

**Syntax**

```
sqlany_get_next_result ( $stmt )
```

**Parameters**

- **\$stmt** A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

**Returns**

Returns a scalar value that is 1 on success or 0 on failure.

**See also**

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)

**Example**

```
stmt = api.sqlany_prepare(conn, "call two_results()" )
rc = api.sqlany_execute( stmt )
```

```
# Fetch from first result set
rc = api.sqlany_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.sqlany_get_next_result( stmt )
# Fetch from second result set
rc = api.sqlany_fetch_absolute( stmt, 2 )
```

## sqlany\_init function

Initializes the interface.

### Syntax

```
sqlany_init ( )
```

### Returns

Returns a 2-element array that contains 1 on success or 0 on failure as the first element and the Ruby interface version as the second element.

### See also

- [“sqlany\\_fini function” on page 708](#)

### Example

```
# Load the SQLAnywhere gem
begin
  require 'rubygems'
  gem 'sqlanywhere'
  unless defined? SQLAnywhere
    require 'sqlanywhere'
  end
end
# Create an interface
api = SQLAnywhere::SQLAnywhereInterface.new()
# Initialize the interface (loads the DLL/SO)
SQLAnywhere::API.sqlany_initialize_interface( api )
# Initialize our api object
api.sqlany_init()
```

## sqlany\_new\_connection function

Creates a connection object.

### Syntax

```
sqlany_new_connection ( )
```

### Returns

Returns a scalar value that is a connection object.

## Remarks

A connection object must be created before a database connection is established. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time.

## See also

- [“sqlany\\_connect function” on page 702](#)
- [“sqlany\\_disconnect function” on page 704](#)

## Example

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Status=#{status}\n"
```

## sqlany\_num\_cols function

Returns number of columns in the result set.

### Syntax

```
sqlany_num_cols ( $stmt )
```

### Parameters

- **\$stmt** A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

### Returns

Returns a scalar value that is the number of columns in the result set, or -1 on a failure.

## See also

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)
- [“sqlany\\_prepare function” on page 716](#)

## Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

## sqlany\_num\_params function

Returns the number of parameters that are expected for a prepared statement.

### Syntax

```
sqlany_num_params ( $stmt )
```

**Parameters**

- **\$stmt** A statement object returned by the successful execution of `sqlany_prepare`.

**Returns**

Returns a scalar value that is the number of parameters in a prepared statement, or -1 on a failure.

**See also**

- [“sqlany\\_prepare function” on page 716](#)

**Example**

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
num_params = api.sqlany_num_params( stmt )
```

## sqlany\_num\_rows function

Returns the number of rows in the result set.

**Syntax**

```
sqlany_num_rows ( $stmt )
```

**Parameters**

- **\$stmt** A statement object executed by `sqlany_execute` or `sqlany_execute_direct`.

**Returns**

Returns a scalar value that is the number of rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

**Remarks**

By default, this function only returns an estimate. To return an exact count, set the `ROW_COUNTS` option on the connection. For more information, see [“row\\_counts option” \[SQL Anywhere Server - Database Administration\]](#).

A count of the number of rows in a result set can be returned only for the first result set in a statement that returns multiple result sets. If `sqlany_get_next_result` is used to move to the next result set, `sqlany_num_rows` will still return the number of rows in the first result set.

**See also**

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_execute\\_direct function” on page 705](#)

### Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

## sqlany\_prepare function

Prepares the supplied SQL string.

### Syntax

```
sqlany_prepare ( $conn, $sql )
```

### Parameters

- **\$conn** A connection object with a connection established using `sqlany_connect`.
- **\$sql** The SQL statement to be prepared.

### Returns

Returns a scalar value that is the statement object, or nil on failure.

### Remarks

The statement associated with the statement object is executed by `sqlany_execute`. You can use `sqlany_free_stmt` to free the resources associated with the statement object.

### See also

- [“sqlany\\_execute function” on page 705](#)
- [“sqlany\\_free\\_stmt function” on page 709](#)
- [“sqlany\\_num\\_params function” on page 714](#)
- [“sqlany\\_describe\\_bind\\_param function” on page 703](#)
- [“sqlany\\_bind\\_param function” on page 700](#)

### Example

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

## sqlany\_rollback function

Rolls back the current transaction.

### Syntax

```
sqlany_rollback ( $conn )
```

### Parameters

- **\$conn** The connection object on which the rollback operation is to be performed.

### Returns

Returns a scalar value that is 1 on success, 0 on failure.

### See also

- [“sqlany\\_commit function” on page 702](#)

### Example

```
rc = api.sqlany_rollback( conn )
```

## sqlany\_sqlstate function

Retrieves the current SQLSTATE.

### Syntax

```
sqlany_sqlstate ( $conn )
```

### Parameters

- **\$conn** A connection object returned from `sqlany_new_connection`.

### Returns

Returns a scalar value that is the current five-character SQLSTATE.

### See also

- [“sqlany\\_error function” on page 704](#)
- [“SQL Anywhere error messages sorted by SQLSTATE” \[Error Messages\]](#)

### Example

```
sql_state = api.sqlany_sqlstate( conn )
```

## Column types

The following Ruby class defines the column types returned by some SQL Anywhere Ruby functions.

```
class Types
  A_INVALID_TYPE = 0
  A_BINARY      = 1
  A_STRING      = 2
  A_DOUBLE      = 3
  A_VAL64       = 4
  A_UVAL64      = 5
  A_VAL32       = 6
  A_UVAL32      = 7
  A_VAL16       = 8
  A_UVAL16      = 9
end
```

```
A_VAL8      = 10
A_UVAL8     = 11
end
```

## Native column types

The following table defines the native column types returned by some SQL Anywhere functions.

Native type value	Native type
384	DT_DATE
388	DT_TIME
390	DT_TIMESTAMP_STRUCT
392	DT_TIMESTAMP
448	DT_VARCHAR
452	DT_FIXCHAR
456	DT_LONGVARCHAR
460	DT_STRING
480	DT_DOUBLE
482	DT_FLOAT
484	DT_DECIMAL
496	DT_INT
500	DT_SMALLINT
524	DT_BINARY
528	DT_LONGBINARY
600	DT_VARIABLE
604	DT_TINYINT
608	DT_BIGINT
612	DT_UNSYNINT
616	DT_UNSSMALLINT



<b>Native type value</b>	<b>Native type</b>
620	DT_UNSBIGINT
624	DT_BIT
628	DT_NSTRING
632	DT_NFIXCHAR
636	DT_NVARCHAR
640	DT_LONGNVARCHAR

---

---

# Sybase Open Client support

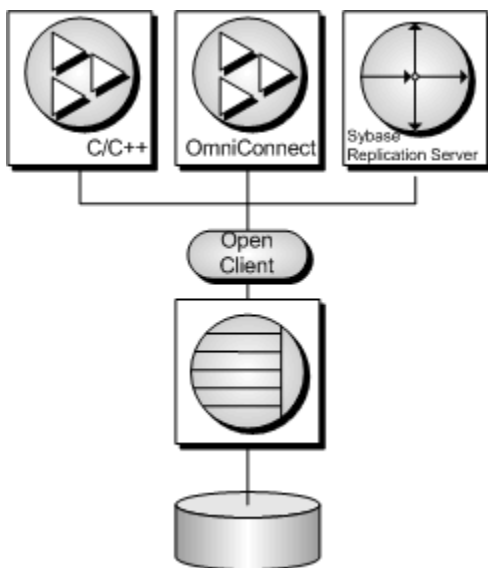
Sybase Open Client provides customer applications, third-party products, and other Sybase products with the interfaces needed to communicate with SQL Anywhere and other Open Servers.

## When to use Open Client

You should consider using the Open Client interface if you are concerned with Adaptive Server Enterprise compatibility or if you are using other Sybase products that support the Open Client interface.

## Open Client applications

You can develop applications in C or C++, and then connect those applications to SQL Anywhere using the Open Client API. Other Sybase applications, such as OmniConnect, use Open Client. The Open Client API is also supported by Sybase Adaptive Server Enterprise.



## See also

- [“SQL Anywhere as an Open Server” \[SQL Anywhere Server - Database Administration\]](#)

## Open Client architecture

### Note

This section describes the Sybase Open Client programming interface for SQL Anywhere. The primary documentation for Sybase Open Client application development is the Open Client documentation, available from Sybase. This section describes features specific to SQL Anywhere, but it is not an exhaustive guide to Sybase Open Client application programming.

Sybase Open Client has two components: programming interfaces and network services.

### DB-Library and Client Library

Sybase Open Client provides two core programming interfaces for writing client applications: DB-Library and Client-Library.

Open Client DB-Library provides support for older Open Client applications, and is a completely separate programming interface from Client-Library. DB-Library is documented in the *Open Client DB-Library/C Reference Manual*, provided with the Sybase Open Client product.

Client-Library programs also depend on CS-Library, which provides routines that are used in both Client-Library and Server-Library applications. Client-Library applications can also use routines from Bulk-Library to help high-speed data transfer.

Both CS-Library and Bulk-Library are included in the Sybase Open Client, which is available separately.

### Network services

Open Client network services include Sybase Net-Library, which provides support for specific network protocols such as TCP/IP and DECnet. The Net-Library interface is invisible to application developers. However, on some platforms, an application may need a different Net-Library driver for different system network configurations. Depending on your host platform, the Net-Library driver is specified either by the system's Sybase configuration or when you compile and link your programs.

Instructions for driver configuration can be found in the *Open Client/Server Configuration Guide*.

Instructions for building Client-Library programs can be found in the *Open Client/Server Programmer's Supplement*.

## What you need to build Open Client applications

To run Open Client applications, you must install and configure Sybase Open Client components on the computer where the application is running. You may have these components present as part of your installation of other Sybase products or you can optionally install these libraries with SQL Anywhere, subject to the terms of your license agreement.

Open Client applications do not need any Open Client components on the computer where the database server is running.

To build Open Client applications, you need the development version of Open Client, available from Sybase.

By default, SQL Anywhere databases are created as case-insensitive, while Adaptive Server Enterprise databases are case sensitive.

### See also

- [“SQL Anywhere as an Open Server” \[SQL Anywhere Server - Database Administration\]](#)

## Data type mappings

Sybase Open Client has its own internal data types, which differ in some details from those available in SQL Anywhere. For this reason, SQL Anywhere internally maps some data types between those used by Open Client applications and those available in SQL Anywhere.

To build Open Client applications, you need the development version of Open Client. To use Open Client applications, the Open Client run-times must be installed and configured on the computer where the application runs.

The SQL Anywhere server does not require any external communications runtime to support Open Client applications.

Each Open Client data type is mapped onto the equivalent SQL Anywhere data type. All Open Client data types are supported.

### SQL Anywhere data types with no direct counterpart in Open Client

The following table lists the mappings of data types supported in SQL Anywhere that have no direct counterpart in Open Client.

SQL Anywhere data type	Open Client data type
unsigned short	int
unsigned int	bigint
unsigned bigint	numeric(20,0)
string	varchar
timestamp	datetime

## Range limitations in data type mapping

Some data types have different ranges in SQL Anywhere than in Open Client. In such cases, overflow errors can occur during retrieval or insertion of data.

The following table lists Open Client application data types that can be mapped to SQL Anywhere data types, but with some restriction in the range of possible values.

The Open Client data type is usually mapped to a SQL Anywhere data type with a greater range of possible values. As a result, it is possible to pass a value to SQL Anywhere that will be accepted and stored in a database, but that is too large to be fetched by an Open Client application.

Data type	Open Client lower range	Open Client upper range	SQL Anywhere lower range	SQL Anywhere upper range
MONEY	-922 377 203 685 477.5808	922 377 203 685 477.5807	-999 999 999 999 999.9999	999 999 999 999 999.9999
SMALLMONEY	-214 748.3648	214 748.3647	-999 999.9999	-999 999.9999
DATETIME [1]	January 1, 1753	December 31, 9999	January 1, 0001	December 31, 9999
SMALLDATETIME	January 1, 1900	June 6, 2079	January 1, 0001	December 31, 9999

[1] For versions earlier than OpenClient 15.5; otherwise, the full range of dates from 0001-01-01 to 9999-12-31 is supported.

### Example

For example, the Open Client MONEY and SMALLMONEY data types do not span the entire numeric range of their underlying SQL Anywhere implementations. Therefore, it is possible to have a value in a SQL Anywhere column which exceeds the boundaries of the Open Client data type MONEY. When the client fetches any such offending values via SQL Anywhere, an error is generated.

### Timestamps

TIMESTAMP values inserted into or retrieved from SQL Anywhere will have the date portion restricted to January 1, 1753 or later and the time version restricted to 1/300th of a second precision if the client is using Open Client 15.1 or earlier. If, however, the client is using Open Client 15.5 or later, then no restriction will apply to the TIMESTAMP values.

## SQL in Open Client applications

This section provides a very brief introduction to using SQL in Open Client applications, with a particular focus on SQL Anywhere-specific issues.

For a complete description, see the Open Client documentation at <http://www.sybase.com/products/databasemanagement/openserver>.

### See also

- “Using SQL in applications” on page 1

## SQL statement execution

You send SQL statements to a database server by including them in Client Library function calls. For example, the following pair of calls executes a DELETE statement:

```
ret = ct_command(cmd, CS_LANG_CMD,
                "DELETE FROM Employees
                WHERE EmployeeID=105"
                CS_NULLTERM,
                CS_UNUSED);
ret = ct_send(cmd);
```

## Using prepared statements

The `ct_dynamic` function is used to manage prepared statements. This function takes a *type* parameter that describes the action you are taking.

### Use a prepared statement in Open Client

1. Prepare the statement using the `ct_dynamic` function, with a `CS_PREPARE` *type* parameter.
2. Set statement parameters using `ct_param`.
3. Execute the statement using `ct_dynamic` with a `CS_EXECUTE` *type* parameter.
4. Free the resources associated with the statement using `ct_dynamic` with a `CS_DEALLOC` *type* parameter.

For more information about using prepared statements in Open Client, see your Open Client documentation.

## Cursors and the Open Client interface

The `ct_cursor` function is used to manage cursors. This function takes a *type* parameter that describes the action you are taking.

### Supported cursor types

Not all the types of cursor that SQL Anywhere supports are available through the Open Client interface. You cannot use scroll cursors, dynamic scroll cursors, or insensitive cursors through Open Client.

Uniqueness and updatability are two properties of cursors. Cursors can be unique (each row carries primary key or uniqueness information, regardless of whether it is used by the application) or not. Cursors can be read-only or updatable. If a cursor is updatable and not unique, performance may suffer, as no prefetching of rows is done in this case, regardless of the `CS_CURSOR_ROWS` setting.

## The steps in using cursors

In contrast to some other interfaces, such as embedded SQL, Open Client associates a cursor with a SQL statement expressed as a string. Embedded SQL first prepares a statement and then the cursor is declared using the statement handle.

### Use cursors in Open Client

1. To declare a cursor in Open Client, use `ct_cursor` with `CS_CURSOR_DECLARE` as the *type* parameter.
2. After declaring a cursor, you can control how many rows are prefetched to the client side each time a row is fetched from the server by using `ct_cursor` with `CS_CURSOR_ROWS` as the *type* parameter.

Storing prefetched rows at the client side reduces the number of calls to the server and this improves overall throughput and turnaround time. Prefetched rows are not immediately passed on to the application; they are stored in a buffer at the client side ready for use.

The setting of the `prefetch` database option controls prefetching of rows for other interfaces. It is ignored by Open Client connections. The `CS_CURSOR_ROWS` setting is ignored for non-unique, updatable cursors.

3. To open a cursor in Open Client, use `ct_cursor` with `CS_CURSOR_OPEN` as the *type* parameter.
4. To fetch each row in to the application, use `ct_fetch`.
5. To close a cursor, you use `ct_cursor` with `CS_CURSOR_CLOSE`.
6. In Open Client, you also need to deallocate the resources associated with a cursor. You do this by using `ct_cursor` with `CS_CURSOR_DEALLOC`. You can also use `CS_CURSOR_CLOSE` with the additional parameter `CS_DEALLOC` to perform these operations in a single step.

## Modifying rows through a cursor

With Open Client, you can delete or update rows in a cursor, as long as the cursor is for a single table. The user must have permissions to update the table and the cursor must be marked for update.

### Modify rows through a cursor

- Instead of carrying out a fetch, you can delete or update the current row of the cursor using `ct_cursor` with `CS_CURSOR_DELETE` or `CS_CURSOR_UPDATE`, respectively.

You cannot insert rows through a cursor in Open Client applications.

## How query results are described in Open Client

Open Client handles result sets in a different way than some other SQL Anywhere interfaces.



In embedded SQL and ODBC, you **describe** a query or stored procedure to set up the proper number and types of variables to receive the results. The description is done on the statement itself.

In Open Client, you do not need to describe a statement. Instead, each row returned from the server can carry a description of its contents. If you use `ct_command` and `ct_send` to execute statements, you can use the `ct_results` function to handle all aspects of rows returned in queries.

If you do not want to use this row-by-row method of handling result sets, you can use `ct_dynamic` to prepare a SQL statement and use `ct_describe` to describe its result set. This corresponds more closely to the describing of SQL statements in other interfaces.

## Known Open Client limitations of SQL Anywhere

Using the Open Client interface, you can use a SQL Anywhere database in much the same way as you would an Adaptive Server Enterprise database. There are some limitations, including the following:

- SQL Anywhere does not support the Adaptive Server Enterprise Commit Service.
- A client/server connection's **capabilities** determine the types of client requests and server responses permitted for that connection. The following capabilities are not supported:
  - CS\_CSR\_ABS
  - CS\_CSR\_FIRST
  - CS\_CSR\_LAST
  - CS\_CSR\_PREV
  - CS\_CSR\_REL
  - CS\_DATA\_BOUNDARY
  - CS\_DATA\_SENSITIVITY
  - CS\_OPT\_FORMATONLY
  - CS\_PROTO\_DYNPROC
  - CS\_REG\_NOTIF
  - CS\_REQ\_BCP
- Security options, such as SSL, are not supported. However, password encryption is supported.
- Open Client applications can connect to SQL Anywhere using TCP/IP.

For more information about capabilities, see the *Open Server Server-Library C Reference Manual*.
- When the CS\_DATAFMT is used with the CS\_DESCRIBE\_INPUT, it does not return the data type of a column when a parameterized variable is sent to SQL Anywhere as input.

---

---

# HTTP web services

## SQL Anywhere as an HTTP web server

SQL Anywhere contains a built-in HTTP web server that allows you to create online web services in SQL Anywhere databases. SQL Anywhere web servers support HTTP and SOAP over HTTP requests sent by web browsers and client applications. The web server performance is optimized because web services are embedded in the database.

SQL Anywhere web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages—such as Perl and Python.

In addition to providing web services over an HTTP web server, SQL Anywhere can function as a SOAP or HTTP client application to access standard web services available over the Internet and other SQL Anywhere HTTP web servers.

### See also

- [“Accessing web services using web clients” on page 766](#)

## Quick start to using SQL Anywhere as an HTTP web server

This section illustrates how to start a SQL Anywhere HTTP web server, create a web service, and access it from a web browser. It does not illustrate SQL Anywhere web service features, such as SOAP over HTTP support and application development, to a full extent. Many SQL Anywhere web service features are available that are beyond the scope of this guide.

The following tasks are performed:

- Start a SQL Anywhere HTTP web server database
- Create a general HTTP web service
- View the web service in a web browser

### Create a SQL Anywhere HTTP web server and a general HTTP web service

1. Start the SQL Anywhere HTTP web server while loading a SQL Anywhere database.

Run the following command at a command prompt:

```
dbeng12 -xs http(port=8082) "%SQLANYSAMP12%\demo.db"
```

**Note**

Use the `dbeng12` command to start a personal database server that can only be accessed by the local host. Use the `dbsrv12` command to start a network database server instead.

The `-xs http(port=8082)` option instructs the server to listen for HTTP requests on port 8082. Use a different port number if a web server is already running on port 8082.

2. Use the `CREATE SERVICE` statement to create a web service that responds to incoming web browser requests.

- a. Connect to the `demo.db` database using Interactive SQL by running the following command:

```
dbisql -c "dbf=%SQLANYAMP12%\demo.db;uid=DBA;pwd=sql"
```

- b. Create a new web service in the database.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE SampleWebService
  TYPE 'web-service-type-clause'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT 'Hello world!';
```

Replace `web-service-type-clause` with the desired web service type. The **HTML** type clause is recommended for web browser compatibility. Other general HTTP web service type clauses include **XML**, **RAW**, and **JSON**.

The `CREATE SERVICE` statement creates the **SampleWebService** web service, which returns the result set of the `SELECT` statement. In this example, the statement returns "Hello world!"

The `AUTHORIZATION OFF` clause indicates that authorization is not required to access the web service.

The `USER DBA` statement indicates that the service statement should be run under the `DBA` login name.

The `AS SELECT` clause allows the service to select from a table or function, or view data directly. Use `AS CALL` as an alternative clause to call a stored procedure.

3. View the web service in a web browser.

On the computer running the SQL Anywhere HTTP web server, open a web browser, such as Internet Explorer or Firefox, and go to the following URL:

```
http://localhost:8082/demo/SampleWebService
```

This URL directs your web browser to the HTTP web server on port 8082. **SampleWebService** prints "Hello world". The result set output is displayed in the format specified by the `web-service-type-clause` from step 2. For more information about how result sets are displayed in web browsers, see ["Web service types" on page 734](#).

**Other sample resources**

Samples are included in the `%SQLANYAMP12%\SQLAnywhere\http` directory.

Other examples might be available on CodeXchange at <http://www.sybase.com/developer/codexchange>.

**See also**

- “Starting an HTTP web server” on page 731
- “Web service types” on page 734
- “CREATE SERVICE statement [HTTP web service]” [*SQL Anywhere Server - SQL Reference*]

## Starting an HTTP web server

The SQL Anywhere HTTP web server starts automatically when you launch the database server with the `-xs dbeng12/dbsrv12` server option at the command line. This server option allows you to perform the following tasks:

- Enable a web service protocol to listen for web service requests.
- Configure network protocol options, such as server port, logging, time-out criteria, and the maximum request size.

**See also**

- “Managing web services in an HTTP web server” on page 734
- “`-xs dbeng12/dbsrv12` server option” [*SQL Anywhere Server - Database Administration*]

## Enabling a web service protocol

Web service protocols are enabled by launching your database server with the `-xs` database server option.

The general format of the command line is as follows:

```
dbsrv12 -xs protocol-type(protocol-options) your-database-name.db
```

Replace *protocol-type* and *protocol-options* with one of the following supported protocols and any appropriate protocol options:

- **HTTP** Use this protocol to listen for HTTP connections. Here is an example.

```
dbsrv12 -xs HTTP(PORT=8082) services.db
```

- **HTTPS** Use this protocol to listen for HTTPS connections. SSL version 3.0 and TLS version 1.0 are supported. Here is an example.

```
dbsrv12 -xs HTTPS(FIPS=N;PORT=8082;IDENTITY=c:\sa12\bin64\rsaserver.id;IDENTITY_PASSWORD=test) services.db
```

**Note**

Network protocol options are available for each supported protocol. These options allow you to control protocol behavior and can be configured at the command line when you launch your database server.

**See also**

- “`-xs dbeng12/dbsrv12` server option” [*SQL Anywhere Server - Database Administration*]
- “Configuration of network protocol options” on page 732

## Configuration of network protocol options

Network protocol options are optional settings that provide control over a specified web service protocol. These settings are configured at the command line when you launch your database server with the `-xs` database server option. See “`-xs dbeng12/dbsrv12 server option`” [*SQL Anywhere Server - Database Administration*].

For example, the following command line configures an HTTPS listener with the `PORT`, `FIPS`, `Identity`, and `Identity_Password` network protocol options specified:

```
dbsrv12 -xs https(PORT=544;FIPS=YES;
    IDENTITY=certificate.id;IDENTITY_PASSWORD=password) your-database-name.db
```

This command starts a database server that enables the HTTPS web service protocol for the `your-database-name.db` database. The network protocol options indicate that the web server should perform the following tasks:

- Listen on port 544 instead of the default HTTPS port.
- Enable FIPS-approved security algorithms to encrypt communications.
- Locate the specified identity file, `certificate.id`, which contains a public certificate and its private key.
- Validate the private key against the specified identity password, `password`.

The following list identifies the network protocol options that are commonly used for web service protocols:

Network protocol option	Available web service protocols	Description
“DatabaseName (DBN) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Specifies the name of a database to use when processing web requests, or uses the <code>REQUIRED</code> or <code>AUTO</code> keyword to specify whether database names are required as part of the URL.
“FIPS protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTPS	Enables FIPS-approved security algorithms to encrypt database files, communications for database client/server communication, and web services.
“Identity protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTPS	Specifies the name of an identity file to use for secure HTTPS connections.
“Identity_Password protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTPS	Specifies the password for the encryption certificate.

Network protocol option	Available web service protocols	Description
“LocalOnly (LO) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Allows a client to choose to connect only to a server on the local computer, if one exists.
“LogFile (LOG) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Specifies the name of the file where the database server writes information about web requests.
“LogFormat (LF) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Controls the format of messages written to the log file where the database server writes information about web requests, and specifies which fields appear in the messages.
“LogOptions (LOPT) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Specifies the types of messages that are recorded in the log where the database server writes information about web requests.
“ServerPort (PORT) protocol option” [ <i>SQL Anywhere Server - Database Administration</i> ]	HTTP, HTTPS	Specifies the port the database server is running on.

**See also**

- “Network protocol options” [*SQL Anywhere Server - Database Administration*]

**Starting multiple HTTP web servers**

A multiple HTTP web server configuration allows you to create web services across databases and have them appear as part of a single web site. You can start multiple HTTP web servers by using multiple instances of the `-xs dbeng12/dbsrv12` server option. This task is performed by specifying a unique port number for each HTTP web server.

**Example**

In this example, the following command line starts two HTTP web services—one for *your-first-database.db* and one for *your-second-database.db*:

```
dbsrv12 -xs http(port=80;dbn=your-first-database),http(port=8800;dbn=your-second-database)
your-first-database.db your-second-database.db
```

**See also**

- “DatabaseName (DBN) protocol option” [[SQL Anywhere Server - Database Administration](#)]
- “-xs dbeng12/dbsrv12 server option” [[SQL Anywhere Server - Database Administration](#)]

## Managing web services in an HTTP web server

Web services refer to software that assists inter-computer data transfer and interoperability. They make segments of business logic available over the Internet. URLs become available to clients when managing web services in an HTTP web server. The conventions used when specifying a URL determine how the server should communicate with web clients.

Web service management involves the following tasks:

- Choosing the types of web services that you want to manage.
- Creating and maintaining those web services

Web services can be created and stored in a SQL Anywhere database.

## Web service types

When a web browser or client application makes a web service request to a SQL Anywhere web service, the request is processed and a result set is returned in the response. SQL Anywhere supports several web service types that provide control over the result set format and how result sets are returned. You specify the web server type with the TYPE clause of the CREATE SERVICE or ALTER SERVICE statement after choosing an appropriate web service type.

The following web service types are supported:

- **HTML** The result set of a statement, function, or procedure is formatted into an HTML document that contains a table. Web browsers display the body of the HTML document.
- **XML** The result set of a statement, function, or procedure is returned as an XML document. Non-XML formatted result sets are automatically formatted into XML. Web browsers display the raw XML code, including tags and attributes.

The XML formatting is the equivalent of using the FOR XML RAW clause in a SELECT statement, such as in the following SQL statement example:

```
SELECT * FROM table-name FOR XML RAW
```

- **RAW** The result set of a statement, function, or procedure is returned without automatic formatting.

This service type provides the most control over the result set. However, you must generate the response by writing the necessary markup (HTML, XML) explicitly within your stored procedure. You can use the SA\_SET\_HTTP\_HEADER system procedure to set the HTTP Content-Type header to specify the MIME type, allowing web browsers to correctly display the result set.



For an example of a stored procedure that works in conjunction with the RAW web service type, see “Customizing web pages” on page 744.

- **JSON** The result set of a statement, function, or procedure is returned in JSON (JavaScript Object Notation). JSON is more compact than XML format and has a similar structure. For more information about JSON, see <http://www.json.org/>.

This service is used by AJAX to make HTTP calls to web applications. For an example of the JSON type, see `%SQLANYSAMP12%\SQLAnywhere\HTTP\json_sample.sql`.

- **SOAP** The result set of a statement, function, or procedure is returned as a SOAP response. SOAP services provide a common data interchange standard to provide data access to disparate client applications that support SOAP. SOAP request and response envelopes are transported as an XML payload using HTTP (SOAP over HTTP). A request to a SOAP service must be a valid SOAP request, not a general HTTP request. The output of SOAP services can be adjusted using the FORMAT and DATATYPE attributes of the CREATE or ALTER SERVICE statement.
- **DISH** A DISH service (Determine SOAP Handler) is a SQL Anywhere SOAP endpoint. The DISH service exposes the WSDL (Web Services Description Language) document that describes all SOAP Operations (SQL Anywhere SOAP services) accessible through it. A SOAP client toolkit builds the client application with interfaces based on the WSDL. The SOAP client application directs all SOAP requests to the SOAP endpoint (the SQL Anywhere DISH service).

## Example

The following example illustrates the creation of a general HTTP web service that uses the **RAW** service type:

```
CREATE PROCEDURE sp_echotext(str LONG VARCHAR)
BEGIN
    CALL sa_set_http_header( 'Content-Type', 'text/plain' );
    SELECT str;
END;

CREATE SERVICE SampleWebService
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_echotext ( :str );
```

## See also

- “Creating or altering a web service” on page 736
- “Web services system procedures” [*SQL Anywhere Server - SQL Reference*]
- “HTTP and SOAP request structures” on page 801
- “CREATE SERVICE statement [HTTP web service]” [*SQL Anywhere Server - SQL Reference*]
- “Creating DISH services” on page 738
- “Creating SOAP over HTTP services” on page 737

## Web service maintenance

Web service maintenance involves the following tasks:

- **Creating or altering web services** Create or alter web services to provide web applications supporting a web browser interface and provide data interchange over the web using REST and SOAP methodologies. For an example of general HTTP web service creation, see [“Quick start to using SQL Anywhere as an HTTP web server” on page 729](#).
- **Dropping web services** Dropping a web service causes the subsequent requests made for that service to return a 404 Not Found HTTP status message. All unresolved requests, intended or unintended, are processed if a **root** web service exists.
- **Commenting on web services** Commenting is optional and allows you to provide documentation for your web services.
- **Creating and customizing a root web service** You can create a **root** web service to handle HTTP requests that do not match any other web service requests.
- **Enabling and disabling web services** A disabled web service returns a 404 Not Found HTTP status message. The METHOD clause specifies the HTTP methods that can be called for a particular web service. See [“CREATE SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#).

## Creating or altering a web service

Creating or altering a web service requires use of the CREATE SERVICE or ALTER SERVICE statement, respectively. This section illustrates how to execute these statements with Interactive SQL to create different kinds of web services. The examples in this section assume that you have connected to a SQL Anywhere database, *your-sql-anywhere-database.db*, through Interactive SQL using the following command:

```
dbisql -c "dbf=your-sql-anywhere-database.db;uid=your-userid;pwd=your-password"
```

### Creating HTTP web services

HTTP web services are classified as HTML, XML or RAW. All HTTP web services can be created or altered using the same CREATE SERVICE and ALTER SERVICE statement syntax.

#### Example

Execute the following statement in Interactive SQL to create a sample general HTTP web service in the HTTP web server:

```
CREATE SERVICE SampleWebService
  TYPE 'web-service-type-clause'
  URL OFF
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

The CREATE SERVICE statement creates a new web service named **SampleWebService** and returns the result set of *sql-statement*. You can replace *sql-statement* with either a SELECT statement to select data from a table or view directly, or a CALL statement to call a stored procedure in the database. For more

information about creating stored functions and procedures for your web services, see [“Developing web service applications in an HTTP web server” on page 744](#).

Replace *web-service-type-clause* with the desired web service type. Valid clauses for HTTP web services include **HTML**, **XML**, **RAW** and **JSON**.

You can view the generated result set for the **SampleWebService** service by accessing the service in a web browser.

### See also

- [“Web service types” on page 734](#)
- [“CREATE SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Browsing an HTTP web server” on page 762](#)

### Creating SOAP over HTTP services

SOAP is a data interchange standard supported by many development environments. A SOAP payload consists of an XML document, known as a SOAP envelope. A SOAP request envelope contains the SOAP operation (a SOAP service) and specifies all appropriate parameters. The SOAP service parses the request envelope to obtain the parameters and calls or selects a stored procedure or function just as any other service does. The presentation layer of the SOAP service streams the result set back to the client within a SOAP envelope in a predefined format as specified by the DISH service's WSDL. For more information about SOAP standards, see <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

By default, SOAP service parameters and result data are typed as XmlSchema string parameters. DATATYPE ON specifies that input parameters and response data should use TRUE types. Specifying DATATYPE changes the WSDL specification accordingly, so that client SOAP toolkits generate interfaces with the appropriate type of parameters and response objects. For more information about how data types are formatted for client applications, see [“Variables accessed from result sets” on page 787](#).

The FORMAT clause is used to target specific SOAP toolkits with varying capabilities. DNET provides Microsoft .NET client applications to consume a SOAP service response as a System.Data.DataSet object. CONCRETE exposes a more general structure that allows an object-oriented application, such as .NET or Java, to generate response objects that package rows and columns. XML returns the entire response as an XML document, exposing it as a string. Clients can further process the data using an XML parser. The FORMAT clause of the CREATE SERVICE statement supports multiple client application types. For examples of other formats supported by SOAP services, see [“HTTP web service examples” on page 806](#).

#### Note

The DATATYPE clause only pertains to SOAP services (there is no data typing in HTML) The FORMAT clause can be specified for either a SOAP or DISH service. A SOAP service FORMAT specification overrides that of the DISH service.

### Example

Execute the following statement in Interactive SQL to create a SOAP over HTTP service:

```
CREATE SERVICE SampleSOAPService
  TYPE 'SOAP'
  DATATYPE ON
  FORMAT 'CONCRETE'
  USER DBA
  AUTHORIZATION OFF
  AS sql-statement;
```

### See also

- “Web service types” on page 734
- “CREATE SERVICE statement [SOAP web service]” [*SQL Anywhere Server - SQL Reference*]
- “ALTER SERVICE statement [SOAP web service]” [*SQL Anywhere Server - SQL Reference*]
- “Creating DISH services” on page 738

### Creating DISH services

SQL Anywhere allows you to create DISH services that act as SOAP endpoints for groups of SOAP services. DISH services also automatically construct WSDL (Web Services Description Language) documents that allow SOAP client toolkits to generate the interfaces necessary to interchange data with the SOAP services described by the WSDL. SOAP services can be added and removed without requiring maintenance to the DISH services because the current working set of SOAP over HTTP services are always exposed.

### Example

Execute the following SQL statements in Interactive SQL to create sample SOAP and DISH services in the HTTP web server:

```
CREATE SERVICE "Samples/TestSoapOp"
  TYPE 'SOAP'
  DATATYPE ON
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT( ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR )
BEGIN
  SELECT i, f, s;
END;

CREATE SERVICE "dnet_endpoint"
  TYPE 'DISH'
  GROUP "Samples"
  FORMAT 'DNET';
```

The first CREATE SERVICE statement creates a new SOAP service named **Samples/TestSoapOp**.

The second CREATE SERVICE statement creates a new DISH service named **dnet\_endpoint**. The **Samples** portion of the GROUP clause identifies the group of SOAP services to expose. You can view the WSDL document generated by the DISH service. When running your SQL Anywhere web server on a computer, you can access the service using the `http://localhost:port-number/dnet_endpoint` URL, where *port-number* is the port number that the server is running on.

In this example, the SOAP service does not contain a FORMAT clause to indicate a SOAP response format. Therefore, the SOAP response format is dictated by the DISH service, which does not override

the **FORMAT** clause of the SOAP service. This feature allows you to create homogeneous DISH services where each DISH endpoint can serve SOAP clients with varying capabilities.

### Creating homogeneous DISH services

When SOAP service definitions defer the specification of the **FORMAT** clause to the DISH service, a set of SOAP services can be grouped together within a DISH service that defines the format. Multiple DISH services can then expose the same group of SOAP services with a different **FORMAT** specifications. If you expand on the **TestSoapOp** example, you can create another DISH service named **java\_endpoint** using the following SQL statement:

```
CREATE SERVICE "java_endpoint"  
  TYPE 'DISH'  
  GROUP "Samples"  
  FORMAT 'CONCRETE';
```

In this example, the SOAP client receives a response object named **TestSoapOp\_Dataset** when it makes a web service request for the **TestSoapOp** operation through the **java\_endpoint** DISH service. The WSDL can be inspected to compare the differences between **dnet\_endpoint** and **java\_endpoint**. Using this technique, a SOAP endpoint can quickly be constructed to meet the needs of a particular SOAP client toolkit.

### See also

- “Web service types” on page 734
- “Browsing an HTTP web server” on page 762
- “HTTP web service examples” on page 806
- “CREATE SERVICE statement [SOAP web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “ALTER SERVICE statement [SOAP web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “Creating SOAP over HTTP services” on page 737

### Dropping a web service

Dropping a web service causes the subsequent requests made for that service to return a 404 Not Found HTTP status message. All unresolved requests, intended or unintended, are processed if a **root** web service exists.

### Example

Execute the following SQL statement to drop a web service named **SampleWebService**:

```
DROP SERVICE SampleWebService;
```

### See also

- “DROP SERVICE statement” [[SQL Anywhere Server - SQL Reference](#)]
- “Creating and customizing a root web service” on page 740

## Commenting on a web service

Providing documentation for a web service requires use of the `COMMENT ON SERVICE` statement. A comment can be removed by setting the *statement* clause to null.

### Example

For example, execute the following SQL statement to create a new comment on a web service named **SampleWebService**:

```
COMMENT ON SERVICE SampleWebService
  IS "This is a comment on my web service.";
```

### See also

- “`COMMENT` statement” [[SQL Anywhere Server - SQL Reference](#)]

## Creating and customizing a root web service

An HTTP client request that does not match any web service request is processed by the **root** web service if a **root** web service is defined.

The **root** web service provides you with an easy and flexible method to handle arbitrary HTTP requests whose URLs are not necessarily known at the time when you build your application, and to handle unrecognized requests.

### Example

This example illustrates how to use a **root** web service, which is stored in a table within the database, to provide content to web browsers and other HTTP clients. It assumes that you have started a local HTTP web server on a single database and listening on port 80. All scripts are run on the web server.

Connect to the database server through Interactive SQL and execute the following SQL statement to create a **root** web service that passes the url host variable, which is supplied by the client, to a procedure named **PageContent**:

```
CREATE SERVICE root
  TYPE 'RAW'
  AUTHORIZATION OFF
  SECURE OFF
  URL ON
  USER DBA
  AS CALL PageContent(:url);
```

The `URL ON` portion specifies that the full path component is made accessible by an HTTP variable named `URL`.

Execute the following SQL statement to create a table for storing page content. In this example, the page content is defined by its URL, MIME-type, and the content itself.

```
CREATE TABLE Page_Content (
  url          VARCHAR(1024) NOT NULL PRIMARY KEY,
  content_type VARCHAR(128)  NOT NULL,
```

```

    image          LONG VARCHAR NOT NULL
);

```

Execute the following SQL statements to populate the table. In this example, the intent is to define the content to be provided to the HTTP client when the index.html page is requested.

```

INSERT INTO Page_Content
VALUES(
    'index.html',
    'text/html',
    '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;

```

Execute the following SQL statements to implement the **PageContent** procedure, which accepts the url host variable that is passed through to the **root** web service:

```

CREATE PROCEDURE PageContent(IN @url LONG VARCHAR)
RESULT ( html_doc LONG VARCHAR )
BEGIN
    DECLARE @status CHAR(3);
    DECLARE @type   VARCHAR(128);
    DECLARE @image  LONG VARCHAR;

    SELECT content_type, image INTO @type, @image
    FROM Page_Content
    WHERE url = @url;

    IF @image is NULL THEN
        SET @status = '404';
        SET @type = 'text/html';
        SET @image = '<html><body><h1>404 - Page Not Found</h1>'
            || '<p>There is no content located at the URL "'
            || html_encode( @url ) || '" on this server.<p>'
            || '</body></html>';
    ELSE
        SET @status = '200';
    END IF;
    CALL sa_set_http_header( '@HttpStatus', @status );
    CALL sa_set_http_header( 'Content-Type', @type );
    SELECT @image;
END;

```

The **root** web service calls the **PageContent** procedure when a request to the HTTP server does not match any other defined web service URL. The procedure checks if the client-supplied URL matches a url in the **Page\_Content** table. The **SELECT** statement sends a response to the client. If the client-supplied URL was not found in the table, a generic 404 - Page Not Found html page is built and sent to the client.

Some browsers will respond to the 404 status with their own page, so there is no guarantee that the generic page will be displayed.

In the error message, the **HTML\_ENCODE** function is used to encode the special characters in the client-supplied URL.

The **@HttpStatus** header is used to set the status code returned with the request. A 404 status indicates a Not Found error, and a 200 status indicates OK. The **'Content-Type'** header is used to set the content type returned with the request. In this example, the content (MIME) type of the index.html page is text/html.

For more information, see “[sa\\_set\\_http\\_header system procedure](#)” [*SQL Anywhere Server - SQL Reference*].

**See also**

- “Customizing web pages” on page 744
- “Browsing an HTTP web server” on page 762
- “HTML\_ENCODE function [Miscellaneous]” [*SQL Anywhere Server - SQL Reference*]
- “CREATE SERVICE statement [HTTP web service]” [*SQL Anywhere Server - SQL Reference*]

**Web service SQL statements**

The following SQL statements are available to assist with web service development:

Web server related SQL statements	Description
<p>“<a href="#">CREATE SERVICE statement [HTTP web service]</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Creates a new HTTP web service.</p>
<p>“<a href="#">CREATE SERVICE statement [SOAP web service]</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Creates a new SOAP over HTTP or DISH service.</p>
<p>“<a href="#">ALTER SERVICE statement [HTTP web service]</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Alters an existing HTTP web service.</p>
<p>“<a href="#">ALTER SERVICE statement [SOAP web service]</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Alters an existing HTTP over SOAP or DISH service.</p>
<p>“<a href="#">COMMENT statement</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Stores a comment for a database object in the system tables.</p> <p>Use the following syntax to comment on a web service:</p> <pre>COMMENT ON SERVICE 'web-service-name'     IS 'your comments'</pre>
<p>“<a href="#">DROP SERVICE statement</a>” [<i>SQL Anywhere Server - SQL Reference</i>]</p>	<p>Drops a web service.</p>



## Connection pooling for web services

Each database that exposes web services has access to a pool of database connections. The pool is grouped by user name such that all services defined under a given USER clause share the same connection pool group.

A service request executing a query for the first time must go through an optimization phase to establish an execution plan. If the plan can be cached and reused, then the optimization phase can be skipped for subsequent executions. HTTP connection pooling leverages the plan caching in database connections by reusing them whenever possible. Each service maintains its own list of connections to optimize reuse. However, during peak loads, a service can steal least utilized connections from within the same user group of connections.

Over time, a given connection may acquire cached plans that can optimize performance for the execution of a number of services. For more information about cached plan settings, see [“max\\_plans\\_cached option” \[SQL Anywhere Server - Database Administration\]](#).

In a connection pool, an HTTP request for a given service tries to acquire a database connection from a pool. Unlike HTTP sessions, pooled connections are sanitized by resetting the connection scope environment, such as connection scope variables and temporary tables.

Database connections that are pooled within the HTTP connection pool are not counted as connections in use for the purposes of licensing. Connections are counted as licensed connections when they are acquired from the pool. A 503 Service Temporarily Unavailable status is returned when an HTTP request exceeds the licensing restrictions while acquiring a connection from the pool.

Web services can only utilize a connection pool when they are defined with AUTHORIZATION OFF. For more information, see [“CREATE SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#) and [“CREATE SERVICE statement \[SOAP web service\]” \[SQL Anywhere Server - SQL Reference\]](#).

A database connection within a pool is not updated when changes occur to database and connection options.

### See also

- [“http\\_connection\\_pool\\_basesize connection property” \[SQL Anywhere Server - Database Administration\]](#)
- [“http\\_connection\\_pool\\_timeout connection property” \[SQL Anywhere Server - Database Administration\]](#)
- [“Web services connection properties” on page 761](#)

## Developing web service applications in an HTTP web server

This section provides an overview of web page creation and customization. It explains how to develop stored procedures for your HTTP web server. It assumes that you have knowledge of starting SQL Anywhere HTTP web servers and creating web services that call stored procedures.

For detailed examples of web service applications, see the `%SQLANYWHERE%\SQLAnywhere\HTTP` directory.

### See also

- [“Starting an HTTP web server” on page 731](#)
- [“Managing web services in an HTTP web server” on page 734](#)

## Customizing web pages

You must first evaluate the format of the web service invoked by the HTTP web server to customize your web pages. For example, web pages are formatted in HTML when the web service specifies the **HTML** type.

The **RAW** web service type provides the most customization because it requires that web service procedures and functions explicitly require coding to provide the required markup such as HTML or XML. The following tasks must be performed to customize web pages when using the **RAW** type:

- Set the HTTP Content-Type header field to the appropriate MIME type, such as text/html, in the called stored procedure.
- Apply appropriate markup for the MIME type when generating web page output from the called stored procedure.

### Example

The following example illustrates how to create a new web service with the **RAW** type specified:

```
CREATE SERVICE WebServiceName
  TYPE 'RAW'
  AUTHORIZATION OFF
  URL ON
  USER DBA
  AS CALL HomePage( :url );
```

In this example, the web service calls the **HomePage** stored procedure, which is required to define a single URL parameter that receives the PATH component of the URL.

### Setting the Content-Type header field

Use the `sa_set_http_header` system procedure to define the HTTP Content-Type header to ensure that web browsers correctly render the content.

The following example illustrates how to format web page output in HTML using the text/html MIME-type with the sa\_set\_http\_header system procedure:

```
CREATE PROCEDURE HomePage (IN url LONG VARCHAR)
  RESULT (html_doc XML)
  BEGIN
    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    -- Your SQL code goes here.
    ...
  END
```

### Applying tagging conventions of the MIME-type

You must apply the tagging conventions of the MIME-type specified by the Content-Type header in your stored procedure. SQL Anywhere provides several functions that allow you to create tags.

The following example illustrates how to use the XMLCONCAT, and XMLELEMENT functions to generate HTML content, assuming that the sa\_set\_http\_header system procedure is used to set the Content-Type header to the text/html MIME-type:

```
XMLCONCAT(
  CAST('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' AS XML),
  XMLELEMENT(
    'HTML',
    XMLELEMENT(
      'HEAD',
      XMLELEMENT('TITLE', 'My Home Page')
    ),
    XMLELEMENT(
      'BODY',
      XMLELEMENT('H1', 'My home on the web'),
      XMLELEMENT('P', 'Thank you for visiting my web site!')
    )
  )
)
```

Since element content is always escaped unless the data type is XML, the above example uses the CAST function. Otherwise, special characters are escaped (for example, &lt; for <).

For an extensive example of a **RAW** web service that generates a custom web page, see [“Creating and customizing a root web service”](#) on page 740.

### See also

- “Web service types” on page 734
- “sa\_set\_http\_header system procedure” [[SQL Anywhere Server - SQL Reference](#)]
- “Web services functions” on page 760
- “XMLCONCAT function [String]” [[SQL Anywhere Server - SQL Reference](#)]
- “XMLELEMENT function [String]” [[SQL Anywhere Server - SQL Reference](#)]
- “Browsing an HTTP web server” on page 762
- “Functions” [[SQL Anywhere Server - SQL Reference](#)]

## Accessing client-supplied HTTP variables and headers

Variables and headers in an HTTP client request can be accessed using one of the following approaches:

- The web service statement declaration to pass them as host parameters of a stored function and procedure call.
- Calling the `HTTP_VARIABLE`, `NEXT_HTTP_VARIABLE`, `HTTP_HEADER`, `NEXT_HTTP_HEADER` functions in a stored function or procedure.

### See also

- [“Variables supplied to web services” on page 783](#)

## Accessing HTTP variables using host parameters

You can reference client-supplied variables when you pass them as host parameters of a function or procedure call.

### Example

The following example illustrates how to access the host parameters used in a web service named **ShowTable**:

```
CREATE SERVICE ShowTable
  TYPE 'RAW'
  AUTHORIZATION ON
  AS CALL ShowTable( :user_name, :table_name );

CREATE PROCEDURE ShowTable(IN user VARCHAR, IN tblname LONG VARCHAR)
BEGIN
  -- write SQL code utilizing the user and tblname variables here.
END;
```

Service host parameters are mapped in the declaration order of procedure parameters. In the above example, the **user\_name** and **table\_name** host parameters map to the **user** and **tblname** parameters, respectively.

For more information about passing the URL as a host parameter, see [“Browsing an HTTP web server” on page 762](#).

## Accessing HTTP variables and headers using web service functions

The `HTTP_VARIABLE`, `NEXT_HTTP_VARIABLE`, `HTTP_HEADER`, `NEXT_HTTP_HEADER` functions can be used to iterate through the variables and headers supplied by the client.

### Accessing variables using `HTTP_VARIABLE` and `HTTP_NEXT_VARIABLE`

You can iterate through all client-supplied variables using `NEXT_HTTP_VARIABLE` and `HTTP_VARIABLE` functions within your stored procedures.

The `HTTP_VARIABLE` function allows you to get the value of a variable name.

The `NEXT_HTTP_VARIABLE` function allows you to iterate through all variables sent by the client. Pass the `NULL` value when calling it for the first time to get the first variable name. Use the returned

variable name as a parameter to an HTTP\_VARIABLE function call to get its value. Passing the previous variable name to the next\_http\_variable call gets the next variable name. Null is returned when the last variable name is passed.

Iterating through the variable names guarantees that each variable name is returned exactly once but the variable name order may not be the same as the order they appear in the client request.

The following example illustrates how to use the HTTP\_VARIABLE function to retrieve values from parameters supplied in a client request that accesses the **ShowDetail** service:

```
CREATE SERVICE ShowDetail
  TYPE 'HTML'
  URL PATH OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowDetail();

CREATE PROCEDURE ShowDetail()
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

The following example illustrates how to retrieve three attributes from header-field values associated with the **image** variable:

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

Supplying an integer as the second parameter allows you to retrieve additional values. The third parameter allows you to retrieve header-field values from multi-part requests. Supply the name of a header field to retrieve its value.

### Accessing headers using HTTP\_HEADER and NEXT\_HTTP\_HEADER

HTTP request headers can be obtained from a request using the NEXT\_HTTP\_HEADER and HTTP\_HEADER functions.

The HTTP\_HEADER function returns the value of the named HTTP header field.

The NEXT\_HTTP\_HEADER function iterates through the HTTP headers and returns the next HTTP header name. Calling this function with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. NULL is returned when the last header name is called.

The following table lists some common HTTP request headers and typical values:

Header name	Header value
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language	en-us
Accept-Charset	utf-8, iso-8859-5;q=0.8
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2; WOW64; SV1; .NET CLR 2.0.50727)
Host	localhost
Connection	Keep-Alive

The following table lists special headers and typical values:

Header Name	Header Value
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	&id=-123&version=109&lang=en

You can use the @HttpStatus special header to set the status code of the request being processed.

The following example illustrates how to format header names and values into an HTML table.

Create the **ShowHTTPHeaders** web service:

```
CREATE SERVICE ShowHTTPHeaders
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL HTTPHeaderExample();
```

Create a **HTTPHeaderExample** procedure that uses the NEXT\_HTTP\_HEADER function to get the name of the header, then uses the HTTP\_HEADER function to retrieve its value:

```
CREATE PROCEDURE HTTPHeaderExample()
  RESULT ( html_string LONG VARCHAR )
  BEGIN
    declare header_name LONG VARCHAR;
    declare header_value LONG VARCHAR;
    declare header_query LONG VARCHAR;
```

```

declare table_rows XML;
set header_name = NULL;
set table_rows = NULL;
header_loop:
LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
        LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    SET header_query = HTTP_HEADER( '@HttpQueryString' );
    -- Format header name and value into an HTML table row
    SET table_rows = table_rows ||
        XMLELEMENT( name "tr",
            XMLATTRIBUTES( 'left' AS "align",
                'top' AS "valign" ),
            XMLELEMENT( name "td", header_name ),
            XMLELEMENT( name "td", header_value ),
            XMLELEMENT( name "td", header_query ) );

END LOOP;
SELECT XMLELEMENT( name "table",
    XMLATTRIBUTES( '' AS "BORDER",
        '10' AS "CELLPADDING",
        '0' AS "CELLSPACING" ),
    XMLELEMENT( name "thead",
        XMLATTRIBUTES( 'left' AS "align",
            'top' AS "valign" ),
            'Header Name' ),
        XMLELEMENT( name "thead",
            XMLATTRIBUTES( 'left' AS "align",
                'top' AS "valign" ),
                'Header Value' ),
            XMLELEMENT( name "thead",
                XMLATTRIBUTES( 'left' AS "align",
                    'top' AS "valign" ),
                    'HTTP Query String' ),
            table_rows );
END;

```

Access the **ShowHTTPHeaders** in a web browser to see the request headers arranged in an HTML table.

### See also

- “Variables supplied to web services” on page 783
- “HTTP\_VARIABLE function [Web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “NEXT\_HTTP\_VARIABLE function [Web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “HTTP\_HEADER function [Web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “NEXT\_HTTP\_HEADER function [Web service]” [[SQL Anywhere Server - SQL Reference](#)]

## Accessing client-supplied SOAP request headers

Headers in SOAP requests can be obtained using a combination of the `NEXT_SOAP_HEADER` and `SOAP_HEADER` functions. The `NEXT_SOAP_HEADER` function iterates through the SOAP headers included within a SOAP request envelope and returns the next SOAP header name. Calling it with `NULL` causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of

the previous header to the NEXT\_SOAP\_HEADER function. This function returns NULL when called with the name of the last header.

The following example illustrates the SOAP header retrieval:

```
SET hd_key = NEXT_SOAP_HEADER( hd_key );
IF hd_key IS NULL THEN
  -- no more header entries
  LEAVE header_loop;
END IF;
```

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the SOAP request.

The SOAP\_HEADER function returns the value of the named SOAP header field, or NULL if not called from an SOAP service. It is used when processing an SOAP request via a web service. If a header for the given field-name does not exist, the return value is NULL.

The example searches for a SOAP header named Authentication. When it finds this header, it extracts the value for entire SOAP header and the values of the **@namespace** and **mustUnderstand** attributes. The SOAP header value might look something like this XML string:

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">
  <userName pwd="none">
    <first>John</first>
    <last>Smith</last>
  </userName>
</Authentication>
```

For this header, the **@namespace** attribute value would be **CustomerOrderURN**

Also, the **mustUnderstand** attribute value would be 1

The interior of this XML string is parsed with the OPENXML function using an XPath string set to **/\*:Authentication/\*:userName**.

```
SELECT * FROM OPENXML( hd_entry, xpath )
WITH ( pwd LONG VARCHAR '@*:pwd',
      first_name LONG VARCHAR '*:first/text()',
      last_name LONG VARCHAR '*:last/text()' );
```

Using the sample SOAP header value shown above, the SELECT statement would create a result set as follows:

pwd	first_name	last_name
none	John	Smith

A cursor is declared on this result set and the three column values are fetched into three variables. At this point, you have all the information of interest that was passed to the web service.

### Example

The following example illustrates how a web server can process SOAP requests containing parameters, and SOAP headers. The example implements an **addItem** SOAP operation that takes two parameters:



**amount** of type int and **item** of type string. The **sp\_addItems** procedure processes an Authentication SOAP header extracting the first and last name of the user. The values are used to populate a SOAP response Validation header via the **sa\_set\_soap\_header** system procedure. The response is a result of three columns: **quantity**, **item** and **status** with types INT, LONG VARCHAR and LONG VARCHAR respectively.

```
// create the SOAP service
CREATE SERVICE addItems
  TYPE 'SOAP'
  FORMAT 'CONCRETE'
  AUTHORIZATION OFF
  USER DBA
  AS CALL sp_addItems( :amount, :item );

// create SOAP endpoint for related services
CREATE SERVICE itemStore
  TYPE 'DISH'
  AUTHORIZATION OFF
  USER DBA;

// create the procedure that will process the SOAP requests for the addItems
service
CREATE PROCEDURE sp_addItems(count INT, item LONG VARCHAR)
RESULT(quantity INT, item LONG VARCHAR, status LONG VARCHAR)
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
  DECLARE pwd LONG VARCHAR;
  DECLARE first_name LONG VARCHAR;
  DECLARE last_name LONG VARCHAR;
  DECLARE xpath LONG VARCHAR;
  DECLARE authinfo LONG VARCHAR;
  DECLARE namespace LONG VARCHAR;
  DECLARE mustUnderstand LONG VARCHAR;

header_loop:
  LOOP
    SET hd_key = next_soap_header( hd_key );
    IF hd_key IS NULL THEN
      // no more header entries.
      leave header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = soap_header( hd_key );
      SET xpath = '/*: ' || hd_key || '/*:userName';
      SET namespace = soap_header( hd_key, 1, '@namespace' );
      SET mustUnderstand = soap_header( hd_key, 1, 'mustUnderstand' );
      BEGIN
        // parse for the pieces that you are interested in
        DECLARE crsr CURSOR FOR SELECT * FROM
          openxml( hd_entry, xpath )
            WITH ( pwd LONG VARCHAR '@*:pwd',
                  first_name LONG VARCHAR '*:first/text()',
                  last_name LONG VARCHAR '*:last/text()' );
        OPEN crsr;
        FETCH crsr INTO pwd, first_name, last_name;
        CLOSE crsr;
      END;
      // build a response header, based on the pieces from the request
header
      SET authinfo = XMLELEMENT( 'Validation',
        XMLATTRIBUTES(
```

```
        namespace as xmlns,
        mustUnderstand as mustUnderstand ),
        XMLELEMENT( 'first', first_name ),
        XMLELEMENT( 'last', last_name ) );
    CALL sa_set_soap_header( 'authinfo', authinfo);
END IF;
END LOOP header_loop;
// code to validate user/session and check item goes here...
SELECT count, item, 'available';
END;
```

For the client-side implementation of this example, see [“SOAP request header management” on page 777](#) or [“Sample: Handling SOAP headers, parameters, and responses” on page 832](#).

## HTTP session management on an HTTP server

A web application can support sessions in various ways. Hidden fields within HTML forms can be used to preserve client/server data across multiple requests. Alternatively, Web 2.0 techniques, such as an AJAX enabled client-side JavaScript, can make asynchronous HTTP requests based on client state. SQL Anywhere offers the additional capability of preserving a database connection for exclusive use of sessioned HTTP requests.

Any connection scope variables and temporary tables created and altered within the HTTP session are accessible to subsequent HTTP requests that specify the given SessionID. The SessionID can be specified by a GET or POST HTTP request method or specified within an HTTP cookie header. When an HTTP request is received with a SessionID variable, the server checks its session repository for a matching context. If it finds a session, the server utilizes its database connection for processing the request. If the session is in use, it queues the HTTP request and activates it when the session is freed.

The `sa_set_http_option` can be used to create, delete and change session ids.

HTTP sessions require special handling for management of the session criteria. Only one database connection exists for use by a given SessionID, so consecutive client requests for that SessionID are serialized by the server. Up to 16 requests can be queued for a given SessionID. Subsequent requests for the given SessionID are rejected with a 503 `Service Unavailable` status when the session queue is full.

When creating a SessionID for the first time, the SessionID is immediately registered by the system. Subsequent requests that modify or delete the SessionID are only applied when the given HTTP request terminates. This approach promotes consistent behavior if the request processing results in a roll-back or if the application deletes and resets the SessionID.

The current session is deleted and replaced with the pending session when an HTTP request changes the SessionID. The database connection cached by the session is effectively moved to the new session context, and all state data is preserved, such as temporary tables and created variables.

For a complete example of HTTP session usage, see `%SQLANYSAMP12%\SQLAnywhere\HTTP\session.sql`.

**Note**

Stale sessions should be deleted and an appropriate time out should be set to minimize the number of outstanding connections because each client application connection holds a license seat. Connections associated with HTTP sessions maintain their hold on the server database for the duration of the connection.

For more information about licensing in SQL Anywhere, see <http://www.sybase.com/detail?id=1056242>.

**See also**

- “sa\_set\_http\_option system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_set\_http\_header system procedure” [*SQL Anywhere Server - SQL Reference*]
- “CONNECTION\_PROPERTY function [System]” [*SQL Anywhere Server - SQL Reference*]
- “Accessing connection property values” [*SQL Anywhere Server - Database Administration*]

**Creating an HTTP session**

Sessions can be created using the SessionID option in the sa\_set\_http\_option system procedure. The session ID can be defined by any non-null string.

Session state management is supported by URLs and cookies. HTTP sessions can be accessed using HTTP cookies, or through the URL of a GET request or from within the body of a POST (x-www-form-urlencoded) request. For example, the following URL utilizes the **XYZ** database connection when it executes:

```
http://localhost/sa_svc?SESSIONID=XYZ
```

The request is processed as a standard session-less request if an **XYZ** database connection does not exist.

**Example**

The following code illustrates how to create a **RAW** web service that creates and deletes sessions. A connection scope variable named **request\_count** is increment each time an HTTP request is made while specifying a valid SessionID.

```
CREATE SERVICE session
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL sp_session();

CREATE PROCEDURE sp_session()
BEGIN
  DECLARE body LONG VARCHAR;
  DECLARE sesid LONG VARCHAR;
  DECLARE newsesid LONG VARCHAR;
  DECLARE createtm TIMESTAMP;
  DECLARE lastaccesstm TIMESTAMP;

  SELECT CONNECTION_PROPERTY('sessionid') INTO sesid;
  IF HTTP_VARIABLE('delete') IS NOT NULL THEN
    CALL sa_set_http_option( 'SessionID', NULL );
    SET body = '<html><body>Deleted ' || sesid
```

```

        || '</BR><a href="http://localhost/session">Start Again</a>';
    SELECT body;
END IF;
IF sesid = '' THEN
    SET newsesid = set_session();
    CREATE VARIABLE request_count INT;
    SET request_count = 0;

    SET body = '<html><body> Created sessionid ' || newsesid
    || '</BR><a href="http://localhost/session?SessionID=' ||
newsesid || '> Enter into Session</a>';
ELSE
    SELECT CONNECTION_PROPERTY('sessionid') INTO sesid;

    SET request_count = request_count +1;
    SELECT CONNECTION_PROPERTY('sessioncreatetime') INTO createtm;
    SELECT CONNECTION_PROPERTY('sessionlasttime') INTO lastaccesstm;

    SET body = '<html><body>Session ' || sesid || '</BR>'
    || 'created ' || createtm || '</BR>'
    || 'last access ' || lastaccesstm || '</BR>'
    || '<H3>REQUEST COUNT is ' || request_count || '</H3><HR></BR>'
    || '<a href="http://localhost/session?SessionID=' || sesid ||
'"> Enter into Session</a></BR>'
    || '<a href="http://localhost/session?SessionID=' || sesid ||
'&delete"> Delete Session</a>';
END IF;

    SELECT body;
END;

```

**See also**

- [“Using the URL to manage a session” on page 754](#)

**Using the URL to manage a session**

In a URL session state management system, the client application or web browser provides the session ID in a URL.

**Example**

The following example illustrates unique session ID creation within an HTTP web server SQL function where session IDs can be provided by a URL only:

```

CREATE FUNCTION set_session()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
    CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND, tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;

```

The SessionID is represented as an empty string if the session\_id is not defined for the connection, making a sessionless connection.

The `sa_set_http_option` system procedure returns an error if the `session_id` is owned by another HTTP request.

### See also

- “`sa_set_http_option` system procedure” [*SQL Anywhere Server - SQL Reference*]
- “`sa_set_http_header` system procedure” [*SQL Anywhere Server - SQL Reference*]
- “`CONNECTION_PROPERTY` function [System]” [*SQL Anywhere Server - SQL Reference*]
- “Accessing connection property values” [*SQL Anywhere Server - Database Administration*]
- “Detecting an inactive HTTP session” on page 756
- “Deleting an HTTP session or changing the session ID” on page 756
- “Web service error code reference” on page 804

### Using cookies to manage a session

In a cookie session state management system, the client application or web browser provides the session ID in an HTTP cookie header instead of a URL. Cookie session management is supported with the 'Set-Cookie' HTTP response header of the `sa_set_http_header` system procedure.

#### Note

You can not rely on cookie state management when cookies can be disabled in the client application or web browser. Support for both URL and cookie state management is recommended. The URL-supplied session ID is used when session IDs are provided by both the URL and a cookie.

### Example

The following example illustrates unique session ID creation within an HTTP web server SQL function where session IDs can be provided by a URL or a cookie:

```
CREATE FUNCTION set_session_cookie()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
        CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND, tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    CALL sa_set_http_header( 'Set-Cookie',
        'sessionid=' || session_id || ';' ||
        'max-age=60;' ||
        'path=/session;' );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

### See also

- “`sa_set_http_option` system procedure” [*SQL Anywhere Server - SQL Reference*]
- “`sa_set_http_header` system procedure” [*SQL Anywhere Server - SQL Reference*]
- “`CONNECTION_PROPERTY` function [System]” [*SQL Anywhere Server - SQL Reference*]
- “Accessing connection property values” [*SQL Anywhere Server - Database Administration*]
- “Detecting an inactive HTTP session” on page 756
- “Deleting an HTTP session or changing the session ID” on page 756

## Detecting an inactive HTTP session

The SessionCreateTime and SessionLastTime connection properties can be used to determine if the current connection is within a session context. The HTTP request is not running within a session context when either connection property query returns an empty string.

The SessionCreateTime connection property provides a metric of when a given session was created. It is initially defined when the sa\_set\_http\_option system procedure is called to establish the SessionID.

The SessionLastTime connection property provides the time when the last processed session request released the database connection upon termination of the previous request. It is returned as an empty string when the session is first created until the creator request releases the connection.

**Note**

You can adjust the session timeout duration using the http\_session\_timeout option.

### Example

The following example illustrates session detection using the SessionCreateTime and SessionLastTime connection properties:

```
SELECT CONNECTION_PROPERTY( 'sessioncreatetime' ) INTO ses_create;  
SELECT CONNECTION_PROPERTY( 'sessionlasttime' ) INTO ses_last;
```

### See also

- [“CONNECTION\\_PROPERTY function \[System\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Accessing connection property values” \[SQL Anywhere Server - Database Administration\]](#)
- [“Creating an HTTP session” on page 753](#)
- [“Deleting an HTTP session or changing the session ID” on page 756](#)
- [“http\\_session\\_timeout option” \[SQL Anywhere Server - Database Administration\]](#)

## Deleting an HTTP session or changing the session ID

Explicitly dropping a database connection that is cached within a session context causes the session to be deleted. Session deletion in this manner is a cancel operation; any requests released from the session queue are in a canceled state. This action ensures that any outstanding requests waiting on the session are terminated. Similarly, a server or database shutdown cancels all database connections.

A Session can be deleted by setting the SessionID option in the sa\_set\_http\_option system procedure to null or an empty string.

The following code can be used for session deletion:

```
CALL sa_set_http_option( 'SessionID', null );
```

When an HTTP session is deleted or the SessionID is changed, any pending HTTP requests that are waiting on the session queue are released and allowed to run outside of a session context. The pending requests do not reuse the same database connection.

A session ID cannot be set to an existing session ID. Pending requests referring to the old SessionID are released to run as session-less requests when a SessionID has changed. Subsequent requests referring to the new SessionID reuse the same database connection instantiated by the old SessionID.

The following conditions are applied when deleting or changing an HTTP session:

- The behavior differs depending on whether the current request had inherited a session whereby a database connection belonging to a session was acquired, or whether a session-less request had instantiated a new session. If the request began as session-less, then the act of creating or deleting a session occurs immediately. If the request has inherited a session, then a change in the session state, such as deleting the session or changing the SessionID, only occurs after the request terminates and its changes have been committed. The difference in behavior addresses processing anomalies that may occur if a client makes simultaneous requests using the same SessionID.
- Changing a session to a SessionID of the current session (has no pending session) is not an error and has no substantial effect.
- Changing a session to a SessionID in use by another HTTP request is an error.
- Changing a session when a change is already pending results in the pending session being deleted and new pending session being created. The pending session is only activated once the request successfully terminates.
- Changing a session with a pending session back to its original SessionID results in the pending session being deleted without any change to the current session.

#### See also

- “[CONNECTION\\_PROPERTY function \[System\]](#)” [*SQL Anywhere Server - SQL Reference*]
- “[Accessing connection property values](#)” [*SQL Anywhere Server - Database Administration*]
- “[sa\\_set\\_http\\_option system procedure](#)” [*SQL Anywhere Server - SQL Reference*]
- “[Detecting an inactive HTTP session](#)” on page 756

## HTTP session administration

A session created by an HTTP request is immediately instantiated so that any subsequent HTTP requests requiring that session context is queued by the session.

In this example, a local host client can access the session with the specified session ID, session\_63315422814117, running within the database, *dbname*, running the service session\_service with the following URL once the session is created on the server with the sa\_set\_http\_option procedure.

```
http://localhost/dbname/session_service?sessionid=session_63315422814117
```

A web application can require a means to track active session usage within the HTTP web server. Session data can be found using the NEXT\_CONNECTION function call to iterate through the active database connections and checking for session related properties such as SessionID.

The following SQL statements illustrate how to track an active session:

```
CREATE VARIABLE conn_id LONG VARCHAR;
CREATE VARIABLE the_sessionID LONG VARCHAR;
SELECT NEXT_CONNECTION( NULL, NULL ) INTO conn_id;
conn_loop:
  LOOP
    IF conn_id IS NULL THEN
      LEAVE conn_loop;
    END IF;
    SELECT CONNECTION_PROPERTY( 'SessionID', conn_id )
      INTO the_sessionID;
    IF the_sessionID != '' THEN
      PRINT 'conn_id = %1!, SessionID = %2!', conn_id, the_sessionID;
    ELSE
      PRINT 'conn_id = %1!', conn_id;
    END IF;
    SELECT NEXT_CONNECTION( conn_id, NULL ) INTO conn_id;
  END LOOP conn_loop;
PRINT '\n';
```

If you examine the database server messages window, you see data that is similar to the following output:

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

Explicitly dropping a connection that belongs to a session causes the connection to be closed and the session to be deleted. If the connection being dropped is currently active in servicing an HTTP request, the request is marked for deletion and the connection is sent a cancel signal to terminate the request. When the request terminates, the session is deleted and the connection closed. Deleting the session causes any pending requests on that session's queue to be re-queued.

In the event the connection is currently inactive, the session is marked for deletion and re-queued to the beginning of the session timeout queue. The session and the connection are deleted in the next timeout cycle (normally within 5 seconds). Any session marked for deletion cannot be used by a new HTTP request.

All sessions are lost when the database is stopped.

### See also

- [“sa\\_set\\_http\\_option system procedure” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Creating an HTTP session” on page 753](#)
- [“Deleting an HTTP session or changing the session ID” on page 756](#)
- [“NEXT\\_CONNECTION function \[System\]” \[SQL Anywhere Server - SQL Reference\]](#)

## HTTP session error codes

The 503 `Service Unavailable` error occurs when a new request tries to access a session where more than 16 requests are pending on that session, or an error occurred while queuing the session.

The 403 `Forbidden` error occurs when the client IP address or host name does not match that of the creator of the session.



A request stipulating a session that does not exist does not implicitly generate an error. It is up to the web application to detect this condition (by checking `SessionID`, `SessionCreateTime`, or `SessionLastTime` connection properties) and do the appropriate action.

**See also**

- [“Web service error code reference” on page 804](#)

## Character set conversion considerations

Character-set conversion is performed automatically on outgoing result sets of text types by default. Result sets of other types, such as binary objects, are not affected. The character set of the request is converted to the HTTP web server character set, and the result set is converted to the client application character set. The server uses the first suitable character set listed in the request when multiple sets are listed.

Character-set conversion can be enabled or disabled by setting the HTTP option 'CharsetConversion' option of the `sa_set_http_option` system procedure.

The following example illustrates how to turn off automatic character-set conversion:

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

You can use the 'AcceptCharset' option of the `sa_set_http_option` system procedure to specify the character-set encoding preference when character-set conversion is enabled.

The following example illustrates how to specify the web service character set encoding preference to ISO-8859-5, if supported; otherwise, set it to UTF-8:

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

Character sets are prioritized by server preference but the selection also considers the client's Accept-Charset criteria. The most favored character set according to the client that is also specified by this option is used.

**See also**

- [“sa\\_set\\_http\\_option system procedure” \[SQL Anywhere Server - SQL Reference\]](#)

## Web services system procedures

The following system procedures are for use with web services:

- “sa\_http\_header\_info system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_http\_php\_page system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_http\_php\_page\_interpreted system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_http\_variable\_info system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_set\_http\_header system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_set\_http\_option system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sa\_set\_soap\_header system procedure” [*SQL Anywhere Server - SQL Reference*]

### See also

- “Web services functions” [*SQL Anywhere Server - SQL Reference*]
- “SQL Anywhere as an HTTP web server” on page 729
- “-xs dbeng12/dbsrv12 server option” [*SQL Anywhere Server - Database Administration*]

## Web services functions

Web service functions assist the handling of HTTP and SOAP requests within web services.

The following functions are available:

- “HTML\_DECODE function [Miscellaneous]” [*SQL Anywhere Server - SQL Reference*]
- “HTML\_ENCODE function [Miscellaneous]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_BODY function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_DECODE function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_ENCODE function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_HEADER function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_RESPONSE\_HEADER function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “HTTP\_VARIABLE function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “NEXT\_HTTP\_HEADER function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “NEXT\_HTTP\_RESPONSE\_HEADER function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “NEXT\_HTTP\_VARIABLE function [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “NEXT\_SOAP\_HEADER function [SOAP]” [*SQL Anywhere Server - SQL Reference*]
- “SOAP\_HEADER function [SOAP]” [*SQL Anywhere Server - SQL Reference*]

There are also many system procedures available for web services.

### See also

- “SQL Anywhere as an HTTP web server” on page 729
- “-xs dbeng12/dbsrv12 server option” [*SQL Anywhere Server - Database Administration*]
- “Web services system procedures” [*SQL Anywhere Server - SQL Reference*]

## Web services connection properties

Web service connection properties can be database properties that are accessible using the `CONNECTION_PROPERTY` function.

Use the following syntax to store a connection property value from the HTTP server to a local variable in a SQL function or procedure:

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO variable_name;
```

The following is a list of useful runtime HTTP request connection properties that are commonly used for web service applications:

- **HttpServiceName** Returns the service name origin for a web application.
- **AuthType** Returns the type of authentication used when connecting.
- **ServerPort** Returns the database server's TCP/IP port number or 0.
- **ClientNodeAddress** Returns the node for the client in a client/server connection.
- **ServerNodeAddress** Returns the node for the server in a client/server connection.
- **BytesReceived** Returns the number of bytes received during client/server communications.

### See also

- “Accessing connection property values” [[SQL Anywhere Server - Database Administration](#)]
- “HttpServiceName connection property” [[SQL Anywhere Server - Database Administration](#)]
- “AuthType connection property” [[SQL Anywhere Server - Database Administration](#)]
- “ServerPort connection property” [[SQL Anywhere Server - Database Administration](#)]
- “ClientNodeAddress connection property” [[SQL Anywhere Server - Database Administration](#)]
- “ServerNodeAddress connection property” [[SQL Anywhere Server - Database Administration](#)]
- “BytesReceived connection property” [[SQL Anywhere Server - Database Administration](#)]

## Web services options

Web service options control various aspects of HTTP server behavior.

Use the following syntax to set a public option in an HTTP server:

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

The following is a list of options that are commonly used in HTTP servers for application configuration:

- **http\_connection\_pool\_basesize** Specifies the nominal threshold size of database connections.
- **http\_connection\_pool\_timeout** Specifies the maximum duration that an unused connection can be retained in the connection pool.

- **http\_session\_timeout** Specifies the default timeout duration, in minutes, that the HTTP session persists during inactivity.
- **request\_timeout** Controls the maximum time a single request can run.
- **webservice\_namespace\_host** Specifies the hostname to be used as the XML namespace within specification for DISH services.

### See also

- “Database options” [[SQL Anywhere Server - Database Administration](#)]
- “http\_connection\_pool\_basesize option” [[SQL Anywhere Server - Database Administration](#)]
- “http\_connection\_pool\_timeout option” [[SQL Anywhere Server - Database Administration](#)]
- “http\_session\_timeout option” [[SQL Anywhere Server - Database Administration](#)]
- “request\_timeout option” [[SQL Anywhere Server - Database Administration](#)]
- “webservice\_namespace\_host option” [[SQL Anywhere Server - Database Administration](#)]

## Browsing an HTTP web server

Available URL names are defined by how your web services are named and designed. Each web service provides its own set of web content. This content is typically generated by custom functions and procedures in your database, but content can also be generated with a URL that specifies a SQL statement. Alternatively, or in conjunction, you can define the **root** web service, which processes all HTTP requests that are not processed by a dedicated service. The **root** web service would typically inspect the request URL and headers to determine how to process the request.

URLs uniquely specify resources such as html content available through HTTP or secured HTTPS requests. This section explains how to format the URL syntax in your web browser so that you can access the web services defined on your SQL Anywhere HTTP web server.

### Note

The information in this section applies to HTTP web servers that use general HTTP web service types, such as RAW, XML, and HTML, and DISH services. You cannot use a browser to issue SOAP requests. JSON services return result sets for consumption by web service applications using AJAX.

### Syntax

```
{http|https}://host-name[:port-number][/dbn]/service-name[/path-name|?url-query]
```

### Parameters

- **host-name and port-number** Specifies the location of the web server and, optionally, the port number if it is not defined as the default HTTP or HTTPS port numbers. The *host-name* can be the IP address of the computer running the web server. The *port-number* must match the port number used when you started the web server.

See “-xs dbeng12/dbsrv12 server option” [[SQL Anywhere Server - Database Administration](#)] and “ServerPort (PORT) protocol option” [[SQL Anywhere Server - Database Administration](#)].

- **dbn** Specifies the name of a database. This database must be running on the web server and contain web services.

You do not need to specify *dbn* if the web server is running only one database or if the database name was specified for the given HTTP/HTTPS listener of the protocol option.

See “-xs dbeng12/dbsrv12 server option” [*SQL Anywhere Server - Database Administration*] and “DatabaseName (DBN) protocol option” [*SQL Anywhere Server - Database Administration*].

- **service-name** Specifies the name of the web service to access. This web service must exist in the database specified by *dbn*. Slash characters (/) are permitted when you create or alter a web service, so you can use them as part of the *service-name*. SQL Anywhere matches the remainder of the URL with the defined services.

The client request is processed if a *service-name* is not specified and the **root** web service is defined. A 404 Not Found error is returned if the server cannot identify an applicable service to process the request. As a side-effect, if the **root** web service does exist and cannot process the request based on the URL criteria, then it is responsible for generating the 404 Not Found error.

- **path-name** After resolving the service name, the remaining slash delimited path can be accessed by a web service procedure. If the service was created with URL ON, then the whole path is accessible using a designated **URL** HTTP variable. If the service was created with URL ELEMENTS, then each path element can be accessed using designated HTTP variables **URL1** to **URL10**.

Path element variables can be defined as host variables within the parameter declaration of the service statement definition. Alternatively, or additionally, HTTP variables can be accessed from within a stored procedure using the HTTP\_VARIABLE function call.

The following example illustrates the SQL statement used to create a web service where the URL clause is set to ELEMENTS:

```
CREATE SERVICE TestWebService
  TYPE 'HTML'
  URL ELEMENTS
  AUTHORIZATION OFF
  USER DBA
  AS CALL TestProcedure ( :url1, :url2 );
```

This **TestWebService** web service calls a procedure that explicitly references the **url1** and **url2** host variables.

You can access this web service using the following URL, assuming that **TestWebService** is running on the **demo** database from **localhost** through the default port:

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/Assignment3
```

This URL accesses **TestWebService**, which runs **TestProcedure** and assigns the **Assignment1** value to **url1**, and the **Assignment2** value to **url2**. Optionally, **TestProcedure** can access other path elements using the HTTP\_VARIABLE function. For example, the **HTTP\_VARIABLE( 'url3' )** function call returns **Assignment3**.

For more information about URL ELEMENTS and URL-based variables, see “[CREATE SERVICE statement \[HTTP web service\]](#)” [[SQL Anywhere Server - SQL Reference](#)].

- **url-query** An HTTP GET request may follow a path with a query component that specifies HTTP variables. Similarly, the body of a POST request using a standard application/x-www-form-urlencoded Content-Type can pass HTTP variables within the request body. In either case, HTTP variables are passed as name/value pairs where the variable name is delimited from its value with an equals sign. Variables are delimited with an ampersand.

HTTP variables can be explicitly declared as host variables within the parameter list of the service-statement, or accessed using the HTTP\_VARIABLE function from within the stored procedure of the service statement.

For example, the following SQL statement creates a web service that requires two host variables. Host variables are identified with a colon ( : ) prefix.

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

Assuming that **ShowSalesOrderDetail** is running on the **demo** database from **localhost** through the default port, you can access the web service using the following URL:

```
http://localhost/demo/ShowSalesOrderDetail?customer_id=101&product_id=300
```

This URL accesses **ShowSalesOrderDetail** and assigns a value of 101 to **customer\_id**, and a value of 300 to **product\_id**. The resultant output is displayed in your web browser in HTML format.

## Remarks

The web browser prompts for user name and password when required to connect to the server. The browser then base64 encodes the user input within an Authorization request header and resends the request.

If your web service URL clause is set to ON or ELEMENTS, the URL syntax properties of *path-name* and *url-query* can be used simultaneously so that the web service is accessible using one of several different formatting options. When using these syntax properties simultaneously, the *path-name* format must be used first followed by the *url-query* format.

In the following example, this SQL statement creates a web service where the URL clause is set to ON, which defines the **url** variable:

```
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :product_id, :url );
```

The following is a sample list of acceptable URLs that assign a **url** value of 101 and a **product\_id** value of 300:

- `http://localhost:80/demo/ShowSalesOrderDetail2/101?product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2?url=101&product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2?product_id=300&url=101`

When a host variable name is assigned more than once in the context of *path-name* and *url-query*, the last assignment always takes precedence. For example, the following sample URLs assign a **url** value of 101 and a **product\_id** value of 300:

- `http://localhost:80/demo/ShowSalesOrderDetail2/302?url=101&product_id=300`
- `http://localhost:80/demo/ShowSalesOrderDetail2/String?product_id=300&url=101`

### See also

- [“CREATE SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Accessing client-supplied HTTP variables and headers” on page 745](#)
- [“HTTP\\_VARIABLE function \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Creating and customizing a root web service” on page 740](#)

### Example

The following URL syntax is used to access a web service named **gallery\_image** that is running in a database named **demo** on a local HTTP server through the default port, assuming that the **gallery\_image** service is defined with **URL ON**:

```
http://localhost/demo/gallery_image/sunset.jpg
```

The URL appears to request a graphic file in a directory from a traditional web server, but it accesses the **gallery\_image** service with **sunset.jpg** specified as an input parameter for an HTTP web server.

The following SQL statement illustrates how the gallery service could be defined on the HTTP server to accomplish this behavior:

```
CREATE SERVICE gallery_image
  TYPE 'RAW'
  URL ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL gallery_image ( :url );
```

The **gallery\_image** service calls a procedure with the same name, passing the client-supplied URL. For a sample implementation of a **gallery\_image** procedure that can be accessed by this web service definitions, see `%SQLANY%SAMP12%\SQLAnywhere\HTTP\gallery.sql`.

## Accessing web services using web clients

SQL Anywhere can be used as a web client to access web services hosted by a SQL Anywhere web server or third party web servers such as Apache or IIS.

In addition to using SQL Anywhere as a web client, SQL Anywhere web services provide client applications with an alternative to traditional interfaces, such as JDBC and ODBC. They are easily deployed because additional components are not needed, and can be accessed from multi-platform client applications written in a variety of languages, including scripting languages — such as Perl and Python.

For more information, see [“SQL Anywhere as an HTTP web server” on page 729](#).

## Quick start to using SQL Anywhere as a web client

This topic illustrates how to use SQL Anywhere as a web client application to connect to a SQL Anywhere HTTP server and access a general HTTP web service. It does not illustrate SQL Anywhere web client capabilities to a full extent. Many SQL Anywhere web client features are available that are beyond the scope of this topic.

The following tasks are performed:

- Create and start a SQL Anywhere web client.
- Create a procedure that connects to a web service on an HTTP server.
- Perform operations on the result set sent by the HTTP server.

You can develop SQL Anywhere web client applications that connect to any type of online web server, but this guide assumes that you have started a local SQL Anywhere HTTP server on port 8082 and want to connect to a web service named **SampleHTMLService**, created with the following SQL statements:

```
CREATE SERVICE SampleHTMLService
  TYPE 'HTML'
  USER DBA
  AUTHORIZATION OFF
  AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
  RESULT(ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR)
  BEGIN
    SELECT i, f, s;
  END;
```

For more information about setting up an HTTP server, see [“Quick start to using SQL Anywhere as an HTTP web server” on page 729](#).

### Create a SQL Anywhere web client application

1. Run the following command to create a SQL Anywhere client database if one does not already exist:

```
dbinit client-database-name
```



Replace *client-database-name* with a new name for your client database.

2. Run the following command to start the client database:

```
dbsrv12 client-database-name.db
```

3. Run the following command to connect to the client database through Interactive SQL:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=client-database-name"
```

4. Create a new client procedure that connects to the **SampleHTMLService** web service using the following SQL statement:

```
CREATE PROCEDURE client_post(f REAL, i INTEGER, s VARCHAR(16), x
VARCHAR(16))
  URL 'http://localhost:8082/SampleHTMLService'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';
```

5. Execute the following SQL statement to call the client procedure and send an HTTP request to the web server:

```
CALL client_post(3.14, 9, 's varchar', 'x varchar');
```

The HTTP POST request created by **client\_post** looks something like this.

```
POST /SampleHTMLService HTTP/1.0
ASA-Id: ea1746b01cd0472eb4f0729948db60a2
User-Agent: SATest
Accept-Charset: windows-1252, UTF-8, *
Date: Wed, 9 Jun 2010 21:55:01 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 58

&f=3.1400001049041748&i=9&s=s%20varchar&x=x%20varchar
```

The web service **SampleHTMLService** running on the web server extracts the parameter values for **i**, **f**, and **s** from the POST request and passes them as parameters to the **sp\_echo** procedure. Parameter value **x** is ignored. The **sp\_echo** procedure creates a result set which is returned to the web service. It is important to note that agreement in parameter names between the client and the web server is essential for proper mapping.

The web service creates the response which is sent back to the client. The output displayed in Interactive SQL should be similar to the following output:

Attribute	Value
Status	HTTP/1.1 200 OK

Attribute	Value
Body	<pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;/SampleHTMLService&lt;/title&gt;&lt;/head&gt; &lt;body&gt; &lt;h3&gt;/SampleHTMLService&lt;/h3&gt; &lt;table border=1&gt; &lt;tr class="header"&gt;&lt;th&gt;&lt;b&gt;ret_i&lt;/b&gt;&lt;/th&gt; &lt;th&gt;&lt;b&gt;ret_f&lt;/b&gt;&lt;/th&gt; &lt;th&gt;&lt;b&gt;ret_s&lt;/b&gt;&lt;/th&gt; &lt;/tr&gt; &lt;tr&gt;&lt;td&gt;9&lt;/td&gt;&lt;td&gt;3.1400001049041748&lt;/td&gt;&lt;td&gt;s varchar&lt;/td&gt;</pre>
Date	Wed, 09 Jun 2010 21:55:01 GMT
Connection	close
Expires	Wed, 09 Jun 2010 21:55:01 GMT
Content-Type	text/html; charset=windows-1252
Server	SQLAnywhere/12.0.1

**See also**

- [“Developing web service applications in an HTTP web server” on page 744](#)
- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE FUNCTION statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Quick start to accessing a SQL Anywhere HTTP web server

This topic illustrates how to access a SQL Anywhere HTTP web server using two different types of client application — Python and C#. It does not illustrate SQL Anywhere web service application capabilities to a full extent. Many SQL Anywhere web service features are available that are beyond the scope of this topic.

The following tasks are performed:

- Create a procedure that connects to a web service on an HTTP server.
- Perform operations on the result set sent by the HTTP server.

You can develop SQL Anywhere web client applications that connect to any type of online web server, but this guide assumes that you have started a local SQL Anywhere HTTP server on port 8082 and want to connect to a web service named **SampleXMLService**, created with the following SQL statements:

```
CREATE SERVICE SampleXMLService
  TYPE 'XML'
  USER DBA
```

```

AUTHORIZATION OFF
AS CALL sp_echo2(:i, :f, :s);

CREATE PROCEDURE sp_echo2(i INTEGER, f NUMERIC(6,2), s LONG VARCHAR )
RESULT( ret_i INTEGER, ret_f NUMERIC(6,2), ret_s LONG VARCHAR )
BEGIN
    SELECT i, f, s;
END;

```

For more information about setting up an HTTP server, see [“Quick start to using SQL Anywhere as an HTTP web server” on page 729](#).

## Access an XML web service using C# or Python

1. Write code that accesses the **SampleXMLService** web service.

- For C#, use the following code:

```

using System;
using System.Xml;

public class WebClient
{
    static void Main(string[] args)
    {
        XmlTextReader reader = new XmlTextReader(
            "http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name == "row")
                    {
                        Console.Write(reader.GetAttribute("ret_i") +
" ");
                        Console.Write(reader.GetAttribute("ret_s") +
" ");
                        Console.WriteLine(reader.GetAttribute("ret_f"));
                    }
                    break;
            }
        }
        reader.Close();
    }
}

```

Save the code to a file named *DocHandler.cs*.

To compile the program, run the following command at a command prompt:

```
csc /out:DocHandler.exe DocHandler.cs
```

- For Python, use the following code:

```

import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_int = attrs.getValue( 'ret_i' )

```

```
        table_string = attrs.getValue( 'ret_s' )
        table_numeric = attrs.getValue( 'ret_f' )
        print('%s %s %s' % ( table_int, table_string,
table_numeric ))

parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse('http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello')
```

Save the code to a file named *DocHandler.py*.

2. Run the application.

- For C#, run the following command:

```
DocHandler
```

- For Python, run the following command:

```
python DocHandler.py
```

The application displays the following output:

```
5 hello 3.14
```

### See also

- [“Developing web service applications in an HTTP web server” on page 744](#)

## Web client application development

SQL Anywhere databases can act as web client applications to access SQL Anywhere hosted web services or web services hosted on third party web servers. SQL Anywhere web client applications are created by writing stored procedures and functions using configuration clauses, such as the URL clause that specifies the web service target endpoint. Web client procedures do not have a body, but in every other way are used as any other stored procedure. When called, a web client procedure makes an outbound HTTP or SOAP request. A web client procedure is restricted from making an outbound HTTP request to itself; it cannot call a localhost SQL Anywhere web service running on the same database.

For detailed examples of web service applications, see the `%SQLANYSAMPI2%\SQLAnywhere\HTTP` directory.

### See also

- [“Web client SQL statements” on page 782](#)

## Web client function and procedure requirements and recommendations

Web service client procedures and functions require the definition of a URL clause to identify the web service endpoint. A web service client procedure or function has specialized clauses for configuration but is used like any other stored procedure or function in every other respect.

You can use the CREATE PROCEDURE and CREATE FUNCTION statements to create web client functions and procedures to send SOAP or HTTP requests to a web server.

The following list outlines the requirements and recommendations for creating or altering web client functions and procedures. You can specify the following information when creating or altering a web client function or procedure:

- The URL clause, which requires an absolute URL specifying the web service endpoint. (Required)
- The TYPE clause to specify whether the request is HTTP or SOAP over HTTP. (Recommended)
- Ports that are accessible to the client application. (Optional)
- The HEADER clause to specify HTTP request headers. (Optional)
- The SOAPHEADER clause to specify SOAP header criteria within the SOAP request envelope. (Optional. For SOAP requests only)
- The namespace URI. (For SOAP requests only)

### Web client URL clause

You must specify the location of the web service endpoint to make it accessible to your web client function or procedure. The URL clause of the CREATE PROCEDURE and CREATE FUNCTION statements provides the web service URL that you want to access.

#### Specifying an HTTP service URL

Specifying an HTTP scheme within the URL clause configures the procedure or function for non-secure communication using an HTTP protocol.

The following statement illustrates how to create a procedure that sends requests to a web service named **SampleHTMLService** that resides in a database named **dbname** hosted by an HTTP web server located at **localhost** on port 8082:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
  URL 'http://localhost:8082/dbname/SampleHTMLService'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SATest';
```

The database name is only required if the HTTP server hosts more than one database. You can substitute **localhost** with the host name or the IP address of the HTTP server.

For more information about starting an HTTP server with these sample specifications, see [“Starting an HTTP web server” on page 731](#).

### Specifying an HTTPS service URL

Specifying an HTTPS scheme within the URL clause configures the procedure or function for secure communication over Secure Socket Layer (SSL).

Your web client application must have access to an RSA server certificate or the certificate that signed the server certificate to issue a secure HTTPS request. The certificate is required for the client procedure in order to authenticate the server to prevent man-in-the-middle exploits.

Use the CERTIFICATE clause of the CREATE PROCEDURE and CREATE FUNCTION statements to authenticate the server and establish a secure data channel. You can either place the certificate in a file and provide the file name, or provide the entire certificate as a string value; you cannot do both.

The following statement demonstrates how to create a procedure that sends requests to a web service named **SecureHTMLService** that resides in a database named **dbname** in an HTTPS server located at **localhost** on the port 8082:

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
  URL 'HTTPS://localhost:8082/dbname/SecureHTMLService'
  CERTIFICATE 'file=c:\sa12\bin64\rsaserver.id'
  TYPE 'HTTP:POST'
  HEADER 'User-Agent:SAtest';
```

The CERTIFICATE clause in this example indicates that the RSA server certificate is located in the `c:\sa12\bin64\rsaserver.id` file.

#### Note

Specifying HTTPS\_FIPS forces the system to use the FIPS libraries. If HTTPS\_FIPS is specified, but no FIPS libraries are present, non-FIPS libraries are used instead.

For more information about starting an HTTP web server with these sample specifications, see [“Starting an HTTP web server” on page 731](#).

### Specifying a proxy server URL

Some requests need to be sent through a proxy server. Use the PROXY clause of the CREATE PROCEDURE and CREATE FUNCTION statements to specify the proxy server URL and redirect requests to that URL. The proxy server forwards the request to the final destination, obtains the response, and forwards the response back to SQL Anywhere.

#### See also

- [“Web client SQL statements” on page 782](#)

## Web service request types

You can specify the type of client requests to send to the web server when creating a web client function or procedure. The **TYPE** clause of the **CREATE PROCEDURE** and **CREATE FUNCTION** statements formats requests before sending them to the web server.

### Specifying an HTTP request format

Web client functions and procedures send HTTP requests when the specified format in the **TYPE** clause begins with an **HTTP** prefix.

For example, execute the following SQL statement in the web client database to create an HTTP procedure named **PostOperation** that sends HTTP requests to the specified URL:

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:POST';
```

In this example, requests are formatted as HTTP:POST requests, which would produce a request similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/12.0.1.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:02:49 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 12

a=123&b=data
```

### Specifying a SOAP request format

Web client functions and procedures send HTTP requests when the specified format in the **TYPE** clause begins with a **SOAP** prefix.

For example, execute the following statement in the web client database to create a SOAP procedure named **SoapOperation** that sends SOAP requests to the specified URL:

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable CHAR(128))
  URL 'HTTP://localhost:8082/dbname/SampleSoapService'
  TYPE 'SOAP:DOC';
```

In this example, a SOAP:DOC request is sent to the URL when you call this procedure, which would produce a request similar to the following:

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/12.0.1.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:05:13 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 428
SOAPAction: "HTTP://localhost:8082/SoapOperation"
```

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The procedure name appears in the `<m:SoapOperation>` tag within the body. The two parameters to the procedure, `intVariable` and `charVariable`, become `<m:intVariable>` and `<m:charVariable>`, respectively.

By default, the stored procedure name is used as the SOAP operation name when building a SOAP request. Parameter names appear in SOAP envelope tag names. You must reference these names correctly when defining a SOAP stored procedure since the server expects these names in the SOAP request. The SET clause can be used to specify an alternate SOAP operation name for the given procedure. WSDL can be used to read a WSDL from a file or URL specification and generate SQL stub functions or procedures. For all but the simplest cases (for example, a SOAP RPC call returning a single string value), it is recommended that function definitions be used rather than procedures. A SOAP function returns the full SOAP response envelope which can be parsed using OPENXML.

**See also**

- [“Web client SQL statements” on page 782](#)
- [“openxml system procedure” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Variables supplied to web services” on page 783](#)
- [“HTTP and SOAP request structures” on page 801](#)

**Web client ports**

It is sometimes necessary to indicate which ports to use when opening a server connection through a firewall. You can use the CLIENTPORT clause of the CREATE PROCEDURE and CREATE FUNCTION statements to designate port numbers on which the client application communicates using TCP/IP. It is recommended that you not use this feature unless your firewall restricts access to a particular range of ports.

For example, execute the following SQL statement in the web client database to create a procedure named **SomeOperation** that sends requests to the specified URL using one of the ports in the range 5050-5060, or port 5070:

```
CREATE PROCEDURE SomeOperation()
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  CLIENTPORT '5050-5060,5070';
```

It is recommended that you specify a range of port numbers when required. Only one connection is maintained at a time when you specify a single port number; the client application attempts to access all



specified port numbers until it finds one to bind to. After closing the connection, a timeout period of several minutes is initiated so that no new connection can be made to the same server and port.

This feature is similar to setting the ClientPort network protocol option.

### See also

- “Web client SQL statements” on page 782
- “ClientPort (CPORT) protocol option (client side only)” [*SQL Anywhere Server - Database Administration*]

## HTTP request header management

HTTP request headers can be added, changed, or removed with the HEADER clause of the CREATE PROCEDURE and CREATE FUNCTION statements. You suppress an HTTP request header by referencing the name. You add or change an HTTP request header value by placing a colon after the header name following by the value. Header value specifications are optional.

For example, execute the following SQL statement in the web client database to create a procedure named **SomeOperation2** that sends requests to the specified URL that puts restrictions on HTTP request headers:

```
CREATE PROCEDURE SomeOperation2()
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:GET'
  HEADER 'SOAPAction\nDate\nFrom:\nCustomAlias:John Doe';
```

In this example, the **Date** header, which is automatically generated by SQL Anywhere, is suppressed. The **From** header is included but is not assigned a value. A new header named **CustomAlias** is included in the HTTP request and is assigned the value of **John Doe**. The GET request looks similar to the following:

```
GET /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/12.0.1.3600
Accept-Charset: windows-1252, UTF-8, *
From:
Host: localhost:8082
Connection: close
CustomAlias: John Doe
```

Folding of long header values is supported, provided that one or more white spaces immediately follow the `\n`.

The following example illustrates long header value support:

```
CREATE PROCEDURE SomeOperation3()
  URL 'HTTP://localhost:8082/dbname/SampleWebService'
  TYPE 'HTTP:POST'
  HEADER 'heading1: This long value\n is really long for a header.\n
  heading2:shortvalue';
```

The POST request looks similar to the following:

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
```

```
User-Agent: SQLAnywhere/12.0.1.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:26:04 GMT
heading1: This long value is really long for a header.
heading2:shortvalue
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=windows-1252
Content-Length: 0
```

**Note**

You must set the **SOAPAction** HTTP request header to the given SOAP service URI as specified in the WSDL when creating a SOAP function or procedure.

**Automatically generated HTTP request headers**

Modifying automatically generated headers can have unexpected results. The following HTTP request headers should not be modified without precaution:

HTTP header	Description
Accept-Charset	Always automatically generated. Changing or deleting this header may result in unexpected data conversion errors.
ASA-Id	Always automatically generated. This header ensures that the client application does not connect to itself to prevent deadlock.
Authorization	Automatically generated when URL contains credentials. Changing or deleting this header may result in failure of the request. Only BASIC authorization is supported. User and password information should only be included when connecting via HTTPS.
Connection	Connection: close, is always automatically generated. Client applications do not support persistent connections. The connection could hang if changed.
Host	Always automatically generated. HTTP/1.1 servers are required to respond with 400 Bad Request if an HTTP/1.1 client does not provide a Host header.
Transfer-Encoding	Automatically generated when posting a request in chunk mode. Removing this header or deleting the chunked value will result in failure when the client is using CHUNK mode.
Content-Length	Automatically generated when posting a request and not in chunk mode. This header is required to tell the server the content length of the body. If the content length is wrong the connection may hang or data loss could occur.

**See also**

- [“Web client SQL statements” on page 782](#)

## SOAP request header management

A SOAP request header is an XML fragment within a SOAP envelope. While the SOAP operation and its parameters can be thought of as an RPC (Remote Procedure Call), a SOAP request header can be used to transfer meta information within a specific request or response. SOAP request headers transport application metadata such as authorization or session criteria.

The value of a SOAPHEADER clause must be a valid XML fragment that conforms to a SOAP request header entry. Multiple SOAP request header entries can be specified. The stored procedure or function automatically injects the SOAP request header entries within a SOAP header element (**SOAP-ENV:Header**). SOAPHEADER values specify SOAP headers that can be declared as a static constant, or dynamically set using the parameter substitution mechanism. The following is a fragment from a sample SOAP request. It contains two XML headers called **Authentication** and **Session** respectively.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="CustomerOrderURN">
      <userName pwd="none" mustUnderstand="1">
        <first>John</first>
        <last>Smith</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Processing SOAP response headers (returned by the SOAP call) differs for functions and procedures. When using a function, which is the most flexible and recommended approach, the entire SOAP response envelope is received. The response envelope can then be processed using openxml to extract SOAP header and SOAP body data. When using a procedure, SOAP response headers can only be extracted through the use of a substitution parameter that maps to an IN or INOUT variable. A SOAP procedure allows for a maximum of one IN or INOUT parameter.

A web service function must parse the response SOAP envelope to obtain the header entries.

### Examples

The following examples illustrate how to create SOAP procedures and functions that send parameters and SOAP headers. Wrapper procedures are used to populate the web service procedure calls and process the responses. The **soapAddItemProc** procedure illustrates the use of a SOAP web service procedure, the **soapAddItemFunc** function illustrates the use of a SOAP web service function, and the **httpAddItemFunc** function illustrates how a SOAP payload may be passed to an HTTP web service procedure.

The following example illustrates a SOAP client procedure that uses substitution parameters to send SOAP headers. A single INOUT parameter is used to receive SOAP headers. A wrapper stored procedure **addItemProcWrapper** that calls **soapAddItemProc** demonstrates how to send and receive soap headers including parameters.

```

CREATE PROCEDURE soapAddItemProc(amount INT, item LONG VARCHAR,
    INOUT inoutheader LONG VARCHAR, IN inheader LONG VARCHAR)
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP( OP=addItems )'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inoutheader!inheader';

CREATE PROCEDURE addItemProcWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR)
BEGIN
    DECLARE io_header LONG VARCHAR;      // inout (write/read) soap header
    DECLARE resxml LONG VARCHAR;
    DECLARE soap_header_sent LONG VARCHAR;
    DECLARE i_header LONG VARCHAR;      // in (write) only soap header
    DECLARE err int;
    DECLARE crsr CURSOR FOR
        CALL soapAddItemProc( amount, item, io_header, i_header );

    SET io_header = XMLELEMENT( 'Authentication',
        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
        XMLELEMENT('userName', XMLATTRIBUTES(
            'none' as pwd,
            '1' as mustUnderstand ),
            XMLELEMENT( 'first', first_name ),
            XMLELEMENT( 'last', last_name ) ) );
    SET i_header = '<Session xmlns="SomeSession">123456789</Session>';
    SET soap_header_sent = io_header || i_header;
    OPEN crsr;
    FETCH crsr INTO resxml, err;
    CLOSE crsr;

    SELECT resxml, err, soap_header_sent, io_header AS soap_header_received;
END;

/* example call to addItemProcWrapper */
CALL addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );

```

The following example illustrates a SOAP client function that uses substitution parameters to send SOAP headers. An entire SOAP response envelope is returned. SOAP headers can be parsed using openxml. A wrapper function **addItemFuncWrapper** that calls **soapAddItemFunc** demonstrates how to send and receive soap headers including parameters. It also shows how to process the response using openxml.

```

CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR,
    IN inheader1 LONG VARCHAR, IN inheader2 LONG VARCHAR )
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inheader1!inheader2';

CREATE PROCEDURE addItemFuncWrapper(amount INT, item LONG VARCHAR,
    first_name LONG VARCHAR, last_name LONG VARCHAR )
BEGIN
    DECLARE i_header1 LONG VARCHAR;
    DECLARE i_header2 LONG VARCHAR;
    DECLARE res LONG VARCHAR;

```

```

DECLARE ns LONG VARCHAR;
DECLARE xpath LONG VARCHAR;
DECLARE header_entry LONG VARCHAR;
DECLARE localname LONG VARCHAR;
DECLARE namespaceuri LONG VARCHAR;
DECLARE r_quantity int;
DECLARE r_item LONG VARCHAR;
DECLARE r_status LONG VARCHAR;

SET i_header1 = XMLELEMENT( 'Authentication',
                           XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                           XMLELEMENT('userName', XMLATTRIBUTES(
                                       'none' as pwd,
                                       '1' as mustUnderstand ),
                                       XMLELEMENT( 'first', first_name ),
                                       XMLELEMENT( 'last', last_name ) ) );
SET i_header2 = '<Session xmlns="SessionURN">123456789</Session>';

SET res = soapAddItemFunc( amount, item, i_header1, i_header2 );

SET ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        | ' xmlns:mp="urn:iAnywhere-com:sa-xpath-metaprop"
        | ' xmlns:customer="CustomerOrderURN"
        | ' xmlns:session="SessionURN"
        | ' xmlns:tns="http://localhost:8082"></ns>';

// Process headers...
SET xpath = '//SOAP-ENV:Header/*';
BEGIN
    DECLARE crsr CURSOR FOR SELECT * FROM
        openxml( res, xpath, 1, ns )
        WITH ( "header_entry" LONG VARCHAR '@mp:xmltext',
              "localname"     LONG VARCHAR '@mp:localname',
              "namespaceuri" LONG VARCHAR
'@mp:namespaceuri' );
    OPEN crsr;
    FETCH crsr INTO "header_entry", "localname", "namespaceuri";
    CLOSE crsr;
END;

// Process body...
SET xpath = '//tns:row';
BEGIN
    DECLARE crsr1 CURSOR FOR SELECT * FROM
        openxml( res, xpath, 1, ns )
        WITH ( "r_quantity" INT 'tns:quantity/text()',
              "r_item"     LONG VARCHAR 'tns:item/text()',
              "r_status"  LONG VARCHAR 'tns:status/
text()' );
    OPEN crsr1;
    FETCH crsr1 INTO "r_quantity", "r_item", "r_status";
    CLOSE crsr1;
END;

SELECT r_item, r_quantity, r_status, header_entry, localname,
namespaceuri;
END;

/* example call to addItemFuncWrapper */
CALL addItemFuncWrapper( 6, 'shorts', 'Jack', 'Smith' );

```

The following example demonstrates how an HTTP:POST can be used as a transport for an entire SOAP payload. Rather than creating a webservice client SOAP procedure, this approach creates a webservice

HTTP procedure that transports the SOAP payload. A wrapper procedure **addItemHttpWrapper** calls **httpAddItemFunc** to demonstrate the use of the POST function. It shows how to send and receive soap headers including parameters and how to accept the response.

```

CREATE FUNCTION httpAddItemFunc(soapPayload XML)
  RETURNS XML
  URL 'http://localhost:8082/itemStore'
  TYPE 'HTTP:POST:text/xml'
  HEADER 'SOAPAction: "http://localhost:8082/addItems"';

CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
  RESULT(response XML)
  BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
  END;

  /* example call to addItemHttpWrapper */
  CALL addItemHttpWrapper( 7, 'socks' );

```

For the server-side implementation of this example, see [“Accessing client-supplied SOAP request headers” on page 749](#) or [“Sample: Handling SOAP headers, parameters, and responses” on page 832](#).

### Limitations

- Server side SOAP services cannot currently define input and output SOAP header requirements. Therefore SOAP header metadata is not available in the WSDL output of a DISH service. This means that a SOAP client toolkit cannot automatically generate SOAP header interfaces for a SQL Anywhere SOAP service endpoint.
- Soap header faults are not supported.

### SOAP namespace URI requirement

The namespace URI specifies the XML namespace used to compose the SOAP request envelope for the given SOAP operation. The domain component from URL clause is used when the namespace URI is not defined.

The server-side SOAP processor uses this URI to understand the names of the various entities in the message body of the request. The NAMESPACE clause of the CREATE PROCEDURE and CREATE FUNCTION statements specifies the namespace URI.

You may be required to specify a namespace URI before procedure calls succeed. This information is usually explained in the public web server documentation, but you can obtain the required namespace URI from the WSDL available from the web server. You can generate a WSDL by accessing the DISH service if you are trying to communicate with a SQL Anywhere web server.

Generally, the NAMESPACE can be copied from the **targetNamespace** attribute specified at the beginning of the WSDL document within the **wsdl:definition** element. Be careful when including any trailing '/', as they are significant. Secondly, check for a **soapAction** attribute for the given SOAP operation. It should correspond to the **SOAPAction** HTTP header that would be generated as explained in the following paragraphs.

The NAMESPACE clause fulfills two functions. It specifies the namespace for the body of the SOAP envelope, and, if the procedure has TYPE 'SOAP:DOC' specified, it is used as the domain component of the **SOAPAction** HTTP header.

The following example illustrates the use of the NAMESPACE clause:

```
CREATE FUNCTION an_operation(a_parameter LONG VARCHAR)
  RETURNS LONG VARCHAR
  URL 'http://wsdl.domain.com/fictitious.asmx'
  TYPE 'SOAP:DOC'
  NAMESPACE 'http://wsdl.domain.com/'
```

Execute the following SQL statement in Interactive SQL:

```
SELECT an_operation('a_value');
```

The statement generates a SOAP request similar to the following output:

```
POST /fictitious.asmx HTTP/1.0
SOAPAction: "http://wsdl.domain.com/an_operation"
Host: wsdl.domain.com
Content-Type: text/xml
Content-Length: 387
Connection: close

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://wsdl.domain.com/">
  <SOAP-ENV:Body>
    <m:an_operation>
      <m:a_parameter>a_value</m:a_parameter>
    </m:an_operation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The namespace for the prefix 'm' is set to **http://wsdl.domain.com/** and the **SOAPAction** HTTP header specifies a fully qualified URL for the SOAP operation.

The trailing slash is not a requirement for correct operation of SQL Anywhere but it can cause a response failure that is difficult to diagnose. The **SOAPAction** HTTP header is correctly generated regardless of the trailing slash.

When a NAMESPACE is not specified, the domain component from the URL clause is used as the namespace for the SOAP body, and if the procedure is of TYPE 'SOAP:DOC', it is used to generate the HTTP **SOAPAction** HTTP header. If in the above example the NAMESPACE clause is omitted, then **http://wsdl.domain.com** is used as the namespace. The subtle difference is that a trailing slash '/' is not present. Every other aspect of the SOAP request, including the **SOAPAction** HTTP header would be identical to the above example.

The NAMESPACE clause is used to specify the namespace for the SOAP body as described for the SOAP:DOC case above. However, the **SOAPAction** HTTP header is generated with an empty value: SOAPAction: ""

When using the SOAP:DOC request type, the namespace is also used to compose the **SOAPAction** HTTP header.

**See also**

- [“Web service request types” on page 773](#)
- [“Web client SQL statements” on page 782](#)
- [“Creating DISH services” on page 738](#)

**Web client SQL statements**

The following SQL statements are available to assist with web client development:

Web client related SQL statements	Description
<a href="#">“CREATE FUNCTION statement [Web service]”</a> <a href="#">[SQL Anywhere Server - SQL Reference]</a>	Creates a web client function that makes an HTTP or SOAP over HTTP request.
<a href="#">“ALTER FUNCTION statement”</a> <a href="#">[SQL Anywhere Server - SQL Reference]</a>	Modifies a function.
<a href="#">“CREATE PROCEDURE statement [Web service]”</a> <a href="#">[SQL Anywhere Server - SQL Reference]</a>	Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.
<a href="#">“ALTER PROCEDURE statement”</a> <a href="#">[SQL Anywhere Server - SQL Reference]</a>	Modifies a procedure.



## Variables supplied to web services

Variables can be supplied to a web service in various ways depending on the web service type.

Web client applications can supply variables to general HTTP web services using any of the following approaches:

- The suffix of the URL
- The body of an HTTP request

Variables can be supplied to the **SOAP** service type by including them as part of a standard SOAP envelope.

### See also

- [“Web service types” on page 734](#)

## Variables supplied in the URLs to web services

The HTTP web server can manage variables supplied in the URL by web browsers. These variables can be expressed in any of the following conventions:

- Appending them to the end of the URL while dividing each parameter value with a slash (/), such as in the following example:

```
http://localhost/database-name/param1/param2/param3
```

- Defining them explicitly in a URL parameter list, such as in the following example:

```
http://localhost/database-name/?arg1=param1&arg2=param2&arg3=param3
```

- A combination of appending them to the URL and defining them in a parameter list, such as in the following example:

```
http://localhost/database-name/param4/param5?  
arg1=param1&arg2=param2&arg3=param3
```

The web server interpretation of the URL depends on how the web service URL clause is specified. For more information about how to specify the URL clause and how the server interprets URL parameters, see [“Browsing an HTTP web server” on page 762](#).

### See also

- [“CREATE SERVICE statement \[HTTP web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Accessing client-supplied HTTP variables and headers” on page 745](#)
- [“Creating and customizing a root web service” on page 740](#)

## Variables supplied in the body HTTP requests

You can supply variables in the body of an HTTP request by specifying HTTP:POST in the TYPE clause in a web client function or procedure.

By default TYPE HTTP:POST uses application/x-www-form-urlencoded mime type. All parameters are urlencoded and passed within the body of the request. Optionally, if a media type is provided, the request Content-Type header is automatically adjusted to the provided media type and a single parameter value is uploaded within the body of the request.

### Example

The following example assumes that a web service named **XMLService** exists on a **localhost** web server. Set up a SQL Anywhere client database, connect to it through Interactive SQL, and execute the following SQL statement:

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)
    URL 'http://localhost/XMLService'
    TYPE 'HTTP:POST:text/xml';
```

The statement creates a procedure that allows you to send a variable in the body of an HTTP request in text/xml format.

Execute the following SQL statement in Interactive SQL to send an HTTP request to the **XMLService** web service:

```
CALL SendXMLContent('<title>Hello World!</title>');
```

The procedure call assigns a value to the **xmlcode** parameter and sends it to web service.

### See also

- [“Accessing client-supplied HTTP variables and headers” on page 745](#)
- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE FUNCTION statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Variables supplied in SOAP envelopes

You can supply variables in a SOAP envelope using the SET SOAP option of a web client function or procedure to set a SOAP operation.

The following code illustrates how to set a SOAP operation in a web client function:

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC';
```

In this example, the **addItems** is the SOAP operation that contains the **amount** and **item** values, which are passed as parameters to the **soapAddItemFunc** function.

You can send a request by running the following sample script:

```
SELECT soapAddItemFunc(5, 'shirt');
```

A call to the **soapAddItemFunc** function call generates a SOAP envelope that looks similar to the following:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>5</m:amount>
      <m:item>shirt</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As an alternative to the previous approach, you can create your own SOAP payload and send it to the server in an HTTP wrapper.

Variables to SOAP services must be included as part of a standard SOAP request. Values supplied using other methods are ignored.

The following code illustrates how to create an HTTP wrapper procedure that builds a customized SOAP envelope:

```
CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
  DECLARE payload XML;
  DECLARE response XML;

  SET payload =
  '<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

  SET response = httpAddItemFunc( payload );
  /* process response as demonstrated in addItemFuncWrapper */
  SELECT response;
END;
```

The following code illustrates the web client function used to send the request:

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
RETURNS XML
URL 'http://localhost:8082/itemStore'
TYPE 'HTTP:POST:text/xml'
HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

You can send a request by running the following sample script:

```
CALL addItemHttpWrapper( 7, 'socks' );
```

## Retrieving result sets from a web service

The SELECT statement can be used to retrieve values from result sets. Once retrieved, these values can be stored in tables or used to set variables.

### Example

In this example, you assume that the web server returns at least two values in the result set.

Execute the following SQL statement to create a table with the intent of storing the values of the result set:

```
CREATE TABLE StoredResults(  
    Attribute LONG VARCHAR,  
    Value     LONG VARCHAR  
);
```

It is assumed that the keywords, **Attribute** and **Value**, are defined in the HTTP response header fields. A **Body** attribute contains the body of the message, which is an HTML document.

Execute the following SQL statement to create a web client procedure:

```
CREATE PROCEDURE test(IN parm CHAR(128))  
    URL 'HTTP://localhost/test'  
    TYPE 'HTTP';
```

The procedure type is HTTP, so this procedure requests a table as a result set.

Result sets can be inserted into the **StoredResults** table as follows:

```
INSERT INTO StoredResults SELECT * FROM test('Storing into a table')  
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

You can add clauses according to the usual syntax of the SELECT statement. For example, if you want only a specific row of the result set you can add a WHERE clause to limit the results of the SELECT to only one row:

```
SELECT Value  
    FROM test('Calling test for the Status Code')  
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR)  
    WHERE Attribute = 'Status';
```

This statement selects only the status information from the result set. It can be used to verify that the call was successful.

### See also

- [“Web client SQL statements” on page 782](#)
- [“Variables accessed from result sets” on page 787](#)

## Variables accessed from result sets

Web service client calls can be made with stored functions or procedures. If made from a function, the return type must be of a character data type, such as CHAR, VARCHAR, or LONG VARCHAR. The body of the HTTP response is the returned value. No header information is included. Additional information about the request, including the HTTP status information, is returned by procedures. So, procedures are preferred when access to additional information is desired.

### SOAP procedures

The response from a SOAP function is an XML document that contains the SOAP response.

SOAP responses are structured XML documents, so SQL Anywhere, by default, attempts to exploit this information and construct a more useful result set. Each of the top-level tags within the returned response document is extracted and used as a column name. The contents below each of these tags in the subtree is used as the row value for that column.

For example, SQL Anywhere would construct the shown data set given the following SOAP response:

```
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
  <SOAP-ENV:Body>
    <ElizaResponse xmlns:SOAPSDK4="SoapInterop">
      <Eliza>Hi, I'm Eliza. Nice to meet you.</Eliza>
    </ElizaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliza
Hi, I'm Eliza. Nice to meet you.

In this example, the response document is delimited by the <ElizaResponse> tags that appear within the <SOAP-ENV:Body> tags.

Result sets have as many columns as there are top-level tags. This result set only has one column because there is only one top-level tag in the SOAP response. This single top-level tag, Eliza, becomes the name of the column.

### XML processing facilities

Information within XML result sets, including SOAP responses, can be accessed using the OPENXML procedure.

The following example uses the OPENXML procedure to extract portions of a SOAP response. This example uses a web service to expose the contents of the SYSWEBSERVICE table as a SOAP service:

```
CREATE SERVICE get_webservices
  TYPE 'SOAP'
  AUTHORIZATION OFF
```

```

USER DBA
AS SELECT * FROM SYSWEBSERVICE;

```

The following web client function, which must be created in a second SQL Anywhere database, issues a call to this web service. The return value of this function is the entire SOAP response document. The response is in the .NET **DataSet** format because **DNET** is the default SOAP service format.

```

CREATE FUNCTION get_webservices()
  RETURNS LONG VARCHAR
  URL 'HTTP://localhost/get_webservices'
  TYPE 'SOAP:DOC';

```

The following statement illustrates how you can use the OPENXML procedure to extract two columns of the result set. The **service\_name** and **secure\_required** columns indicate which SOAP services are secure and where HTTPS is required.

```

SELECT *
FROM openxml( get_webservices(), '//row' )
WITH ( "Name"      CHAR(128) 'service_name',
      "Secure?"   CHAR(1)   'secure_required' );

```

This statement works by selecting the decedents of the **row** node. The WITH clause constructs the result set based on the two elements of interest. Assuming only the **get\_webservices** web service exists, this function returns the following result set:

Name	Secure?
get_webservices	N

### Other types of procedures

Procedures of other types return all the information about a response in a two-column result set. This result set includes the response status, header information and body. The first column, is named **Attribute** and the second is named **Value**. Both are of data type LONG VARCHAR.

The result set has one row for each of the response header fields, and a row for the HTTP status line (Status attribute) and a row for the response body (Body attribute).

The following example represents a typical response:

Attribute	Value
Status	HTTP /1.0 200 OK
Body	<!DOCTYPE HTML ... ><HTML> ... </HTML>
Content-Type	text/html
Server	GWS/2.1
Content-Length	2234

Attribute	Value
Date	Mon, 18 Oct 2004, 16:00:00 GMT

**See also**

- [“Variables accessed from result sets” on page 787](#)
- [“Use of XML in the database” \[SQL Anywhere Server - SQL Usage\]](#)
- [“Sample: Handling SOAP headers, parameters, and responses” on page 832](#)
- [“openxml system procedure” \[SQL Anywhere Server - SQL Reference\]](#)

**SOAP data types**

By default, the XML encoding of parameter input is string and the result set output for SOAP service formats contains no information that specifically describes the data type of the columns in the result set. For all formats, parameter data types are string. For the DNET format, within the schema section of the response, all columns are typed as string. CONCRETE and XML formats contain no data type information in the response. This default behavior can be manipulated using the DATATYPE clause.

SQL Anywhere enables data typing using the DATATYPE clause. Data type information can be included in the XML encoding of parameter input and result set output or responses for all SOAP service formats. This simplifies parameter passing from SOAP toolkits by not requiring client code to explicitly convert parameters to Strings. For example, an integer can be passed as an int. XML encoded data types enable a SOAP toolkit to parse and cast the data to the appropriate type.

When using string data types exclusively, the application needs to implicitly know the data type for every column within the result set. This is not necessary when data typing is requested of the web server. To control whether data type information is included, the DATATYPE clause can be used when the web service is defined.

Here is an example of a web service definition that enlists data typing for the result set response.

```
CREATE SERVICE "SASoapTest/EmployeeList"
  TYPE 'SOAP'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA
  DATATYPE OUT
  AS SELECT * FROM Employees;
```

In this example, data type information is requested for result set responses only since this service does not have parameters.

Data typing is applicable to all SQL Anywhere web services defined as type 'SOAP'.

**Data typing of input parameters**

Data typing of input parameters is supported by simply exposing the parameter data types as their true data types in the WSDL generated by the DISH service.

A typical string parameter definition (or a non-typed parameter) would look like the following:

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar" nillable="true"
type="s:string" />
```

The String parameter may be nillable, that is, it may or may not occur.

For a typed parameter such as an integer, the parameter must occur and is not nillable. The following is an example.

```
<s:element minOccurs="1" maxOccurs="1" name="an_int" nillable="false"
type="s:int" />
```

### Data typing of output parameters

All SQL Anywhere web services of type 'SOAP' may expose data type information within the response data. The data types are exposed as attributes within the rowset column element.

The following is an example of a typed SimpleDataSet response from a SOAP FORMAT 'CONCRETE' web service.

```
<SOAP-ENV:Body>
  <tns:test_types_concrete_onResponse>
    <tns:test_types_concrete_onResult xsi:type='tns:SimpleDataset'>
      <tns:rowset>
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_concrete_onResult>
    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_concrete_onResponse>
</SOAP-ENV:Body>
```

The following is an example of a response from a SOAP FORMAT 'XML' web service returning the XML data as a string. The interior rowset consists of encoded XML and is presented here in its decoded form for legibility.

```
<SOAP-ENV:Body>
  <tns:test_types_XML_onResponse>
    <tns:test_types_XML_onResult xsi:type='xsd:string'>
      <tns:rowset
        xmlns:tns="http://localhost/satest/dish"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <tns:row>
          <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
          <tns:i xsi:type="xsd:int">99</tns:i>
          <tns:ii xsi:type="xsd:long">99999999</tns:ii>
          <tns:f xsi:type="xsd:float">3.25</tns:f>
          <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
          <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
          <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
        </tns:row>
      </tns:rowset>
    </tns:test_types_XML_onResult>
```



```

    <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_XML_onResponse>
</SOAP-ENV:Body>

```

Note that, in addition to the data type information, the namespace for the elements and the XML schema provides all the information necessary for post processing by an XML parser. When no data type information exists in the result set (DATATYPE OFF or IN) then the xsi:type and the XML schema namespace declarations are omitted.

An example of a SOAP FORMAT 'DNET' web service returning a typed SimpleDataSet follows:

```

<SOAP-ENV:Body>
  <tns:test_types_dnet_outResponse>
    <tns:test_types_dnet_outResult xsi:type='sqlresultstream:SqlRowSet'>
      <xsd:schema id='Schema2'
        xmlns:xsd='http://www.w3.org/2001/XMLSchema'
        xmlns:msdata='urn:schemas-microsoft-com:xml-msdata'>
        <xsd:element name='rowset' msdata:IsDataSet='true'>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name='row' minOccurs='0' maxOccurs='unbounded'>
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name='lvc' minOccurs='0' type='xsd:string' />
                    <xsd:element name='ub' minOccurs='0' type='xsd:unsignedByte' />
                    <xsd:element name='s' minOccurs='0' type='xsd:short' />
                    <xsd:element name='us' minOccurs='0' type='xsd:unsignedShort' />
                    <xsd:element name='i' minOccurs='0' type='xsd:int' />
                    <xsd:element name='ui' minOccurs='0' type='xsd:unsignedInt' />
                    <xsd:element name='l' minOccurs='0' type='xsd:long' />
                    <xsd:element name='ul' minOccurs='0' type='xsd:unsignedLong' />
                    <xsd:element name='f' minOccurs='0' type='xsd:float' />
                    <xsd:element name='d' minOccurs='0' type='xsd:double' />
                    <xsd:element name='bin' minOccurs='0' type='xsd:base64Binary' />
                    <xsd:element name='bool' minOccurs='0' type='xsd:boolean' />
                    <xsd:element name='num' minOccurs='0' type='xsd:decimal' />
                    <xsd:element name='dc' minOccurs='0' type='xsd:decimal' />
                    <xsd:element name='date' minOccurs='0' type='xsd:date' />
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
      <diffgr:diffgram xmlns:msdata='urn:schemas-microsoft-com:xml-msdata'
        xmlns:diffgr='urn:schemas-microsoft-com:xml-diffgram-v1'>
        <rowset>
          <row>
            <lvc>Hello World</lvc>
            <ub>128</ub>
            <s>-99</s>
            <us>33000</us>
            <i>-2147483640</i>
            <ui>4294967295</ui>
            <l>-9223372036854775807</l>
            <ul>18446744073709551615</ul>
            <f>3.25</f>
            <d>.555555555555555582</d>
            <bin>QUJD</bin>
            <bool>1</bool>
            <num>123456.123457</num>
          </row>
        </rowset>
      </diffgram>
    </tns:test_types_dnet_outResult>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>

```

```

        <dc>-1.756000</dc>
        <date>2006-05-29-04:00</date>
    </row>
</rowset>
</diffgr:diffgram>
</tns:test_types_dnet_outResult>
<tns:sqlcode>0</tns:sqlcode>
</tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>

```

**Mapping SQL Anywhere types to XML schema types**

SQL Anywhere type	XML schema type	XML example
CHAR	string	Hello World
VARCHAR	string	Hello World
LONG VARCHAR	string	Hello World
TEXT	string	Hello World
NCHAR	string	Hello World
NVARCHAR	string	Hello World
LONG NVARCHAR	string	Hello World
NTEXT	string	Hello World
UNIQUEIDENTIFIER	string	12345678-1234-5678-9012-123456789012
UNIQUEIDENTIFIERSTR	string	12345678-1234-5678-9012-123456789012
XML	This is user defined. A parameter is assumed to be valid XML representing a complex type (for example, base64Binary, SOAP array, struct).	<inputHexBinary xsi:type="xsd:hexBinary"> 414243 </inputHexBinary>  (interpreted as 'ABC')
BIGINT	long	-9223372036854775807
UNSIGNED BIGINT	unsignedLong	18446744073709551615
BIT	boolean	1
VARBIT	string	11111111
LONG VARBIT	string	00000000000000000100000000000000

SQL Anywhere type	XML schema type	XML example
DECIMAL	decimal	-1.756000
DOUBLE	double	.55555555555555582
FLOAT	float	12.3456792831420898
INTEGER	int	-2147483640
UNSIGNED INTEGER	unsignedInt	4294967295
NUMERIC	decimal	123456.123457
REAL	float	3.25
SMALLINT	short	-99
UNSIGNED SMALLINT	unsignedShort	33000
TINYINT	unsignedByte	128
MONEY	decimal	12345678.9900
SMALLMONEY	decimal	12.3400
DATE	date	2006-11-21-05:00
DATETIME	dateTime	2006-05-21T09:00:00.000-08:00
SMALLDATETIME	dateTime	2007-01-15T09:00:00.000-08:00
TIME	time	14:14:48.980-05:00
TIMESTAMP	dateTime	2007-01-12T21:02:14.420-06:00
TIMESTAMP WITH TIME ZONE	dateTime	2007-01-12T21:02:14.420-06:00
BINARY	base64Binary	AAAAGg==
IMAGE	base64Binary	AAAAGg==
LONG BINARY	base64Binary	AAAAGg==
VARBINARY	base64Binary	AAAAGg==

When one or more parameters are of type NCHAR, NVARCHAR, LONG NVARCHAR, or NTEXT then the response output is in UTF8. If the client database uses the UTF-8 character encoding, there is no change in behavior (since NCHAR and CHAR data types are the same). However, if the database does

not use the UTF-8 character encoding, then all parameters that are not an NCHAR data type are converted to UTF8. The value of the XML declaration encoding and Content-Type HTTP header will correspond to the character encoding used.

**Mapping XML schema types to Java types**

XML schema type	Java data type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object

XML schema type	Java data type
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

## SOAP structured data types

### XML return values

The SQL Anywhere server as a web service client may interface to a web service using a function or a procedure.

A string representation within a result set may suffice for simple return data types. The use of a stored procedure may be warranted in this case.

The use of web service functions are a better choice when returning complex data such as arrays or structures. For function declarations, the RETURN clause can specify an XML data type. The returned XML can be parsed using OPENXML to extract the elements of interest.

Note that a return of XML data such as dateTime will be rendered within the result set verbatim. For example, if a TIMESTAMP column was included within a result set, it would be formatted as an XML dateTime string (2006-12-25T12:00:00.000-05:00) not as a string (2006-12-25 12:00:00.000).

### XML parameter values

The SQL Anywhere XML data type is supported for use as a parameter within web service functions and procedures. For simple types, the parameter element is automatically constructed when generating the SOAP request body. However, for XML parameter types, this cannot be done since the XML representation of the element may require attributes that provide additional data. Therefore, when generating the XML for a parameter whose data type is XML, the root element name must correspond to the parameter name.

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

The XML type demonstrates how to send a parameter as a hexBinary XML type. The SOAP endpoint expects that the parameter name (or in XML terms, the root element name) is "inputHexBinary".

### Cookbook constants

Knowledge of how SQL Anywhere references namespaces is required to construct complex structures and arrays. The prefixes listed here correspond to the namespace declarations generated for a SQL Anywhere SOAP request envelope.

SQL Anywhere XML prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	namespace as defined in the NAMESPACE clause

### Complex data type examples

The following three examples demonstrate how to create web service client functions taking parameters that represent an array, a structure, and an array of structures. The web service functions will communicate to SOAP operations (or RPC function names) named echoFloatArray, echoStruct, and echoStructArray respectively. The common namespace used for Interoperability testing is <http://soapinterop.org/>, allowing a given function to test against alternative Interoperability servers simply by changing the URL clause to the chosen SOAP endpoint.

The examples are designed to issue requests to the Microsoft SOAP ToolKit 3.0 Round 2 Interoperability test server at <http://mssoapinterop.org/stkV3>.

All the examples use a table to generate the XML data. The following shows how to set up that table.

```
CREATE LOCAL TEMPORARY TABLE SoapData
(
    seqno INT DEFAULT AUTOINCREMENT,
    i INT,
    f FLOAT,
    s LONG VARCHAR
) ON COMMIT PRESERVE ROWS;

INSERT INTO SoapData (i,f,s)
VALUES (99,99.999,'Ninety-Nine');

INSERT INTO SoapData (i,f,s)
VALUES (199,199.999,'Hundred and Ninety-Nine');
```

The following three functions send SOAP requests to the Interoperability server. Note that this sample issues requests to the Microsoft Interop server:

```
CREATE FUNCTION echoFloatArray(inputFloatArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStruct(inputStruct XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStructArray(inputStructArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
```

```
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';
```

Finally, the three example statements along with the XML representation of their parameters are presented:

1. The parameters in the following example represent an array.

```
SELECT echoFloatArray(
    XMLELEMENT( 'inputFloatArray',
        XMLATTRIBUTES( 'xsd:float[2]' as "SOAP-ENC:arrayType" ),
        (
            SELECT XMLAGG( XMLELEMENT( 'number', f ) ORDER BY seqno )
            FROM SoapData
        )
    )
);
```

The stored procedure echoFloatArray will send the following XML to the Interoperability server.

```
<inputFloatArray SOAP-ENC:arrayType="xsd:float[2]">
<number>99.9990005493164</number>
<number>199.998992919922</number>
</inputFloatArray>
```

The response from the Interoperability server is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoFloatArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result SOAPSDK3:arrayType="SOAPSDK1:float[2]"
        SOAPSDK3:offset="[0]"
        SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK3:float>99.9990005493164</SOAPSDK3:float>
        <SOAPSDK3:float>199.998992919922</SOAPSDK3:float>
      </Result>
    </SOAPSDK4:echoFloatArrayResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the response was stored in a variable, then it can be parsed using OPENXML.

```
SELECT * FROM openxml( resp,'//*:Result/*' )
WITH ( varFloat FLOAT 'text()' );
```

varFloat
99.9990005493
199.9989929199

2. The parameters in the following example represent a structure.

```
SELECT echoStruct(
    XMLELEMENT('inputStruct',
        (
            SELECT XMLFOREST( s as varString,
                              i as varInt,
                              f as varFloat )
            FROM SoapData
            WHERE seqno=1
        )
    )
);
```

The stored procedure echoStruct will send the following XML to the Interoperability server.

```
<inputStruct>
  <varString>Ninety-Nine</varString>
  <varInt>99</varInt>
  <varFloat>99.9990005493164</varFloat>
</inputStruct>
```

The response from the Interoperability server is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoStructResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result href="#id1"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
      xmlns:SOAPSDK5="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the response was stored in a variable, then it can be parsed using OPENXML.

```
SELECT * FROM openxml( resp, '/*:Body/*:SOAPStruct' )
WITH (
  varString LONG VARCHAR 'varString',
  varInt INT 'varInt',
  varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493

- The parameters in the following example represent an array of structures.

```
SELECT echoStructArray(
    XMLELEMENT( 'inputStructArray',
```



```

XMLATTRIBUTES( 'http://soapinterop.org/xsd' AS "xmlns:q2",
               'q2:SOAPStruct[2]' AS "SOAP-ENC:arrayType" ),
(
  SELECT XMLAGG(
    XMLElement('q2:SOAPStruct',
      XMLFOREST( s as varString,
                i as varInt,
                f as varFloat )
    )
  )
  ORDER BY seqno
)
FROM SoapData
);

```

The stored procedure echoFloatArray will send the following XML to the Interoperability server.

```

<inputStructArray xmlns:q2="http://soapinterop.org/xsd"
  SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
  </q2:SOAPStruct>
  <q2:SOAPStruct>
    <varString>Hundred and Ninety-Nine</varString>
    <varInt>199</varInt>
    <varFloat>199.998992919922</varFloat>
  </q2:SOAPStruct>
</inputStructArray>

```

The response from the Interoperability server is shown below.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:echoStructArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result xmlns:SOAPSDK5="http://soapinterop.org/xsd"
        SOAPSDK3:arrayType="SOAPSDK5:SOAPStruct[2]"
        SOAPSDK3:offset="[0]" SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK5:SOAPStruct href="#id1"/>
        <SOAPSDK5:SOAPStruct href="#id2"/>
      </Result>
    </SOAPSDK4:echoStructArrayResponse>
    <SOAPSDK6:SOAPStruct
      xmlns:SOAPSDK6="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK6:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK6:SOAPStruct>
    <SOAPSDK7:SOAPStruct
      xmlns:SOAPSDK7="http://soapinterop.org/xsd"
      id="id2"

```

```

SOAPSDK3:root="0"
SOAPSDK2:type="SOAPSDK7:SOAPStruct">
  <varString>Hundred and Ninety-Nine</varString>
  <varInt>199</varInt>
  <varFloat>199.998992919922</varFloat>
</SOAPSDK7:SOAPStruct>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

If the response was stored in a variable, then it can be parsed using OPENXML.

```

SELECT * FROM openxml( resp, '//*[@:Body/*:SOAPStruct' )
WITH (
  varString LONG VARCHAR 'varString',
  varInt INT 'varInt',
  varFloat FLOAT 'varFloat' );

```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493
Hundred and Ninety-Nine	199	199.9989929199

## Substitution parameters used for clause values

Declared parameters to a stored procedure or function are automatically substituted for placeholders within a clause definition each time the stored procedure or function is run. Substitution parameters allow the creation of general web service procedures that dynamically configure clauses at run time. Any substrings that contain an exclamation mark '!' followed by the name of one of the declared parameters is replaced by that parameter's value. In this way one or more parameter values may be substituted to derive one or more clause values at runtime.

Parameter substitution requires adherence to the following rules:

- All parameters used for substitution must be alphanumeric. Underscores are not allowed.
- A substitution parameter must be followed immediately by a non-alphanumeric character or termination. For example, **!sizeXL** is not substituted with the value of a parameter named size because X is alphanumeric.
- A substitution parameter that is not matched to a parameter name is ignored.
- An exclamation mark (!) can be escaped with another exclamation mark.

For example, the following procedure illustrates the use of parameter substitution. URL and HTTP header definitions must be passed as parameters.

```

CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG VARCHAR)
  URL 'HTTP://!uid:pwd@localhost/myservice'
  HEADER '!headers';

```

You can then use the following statement to call the **test** procedure and initiate an HTTP request:

```
CALL test('dba', 'sql', 'NewHeader1:value1\nNewHeader2:value2');
```

Different values can be used each time this procedure is called.

### Encryption certificate example

You can use parameter substitution to pass encryption certificates from a file and pass them to a stored procedure or stored function.

The following example illustrates how to pass a certificate as a substitution string:

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Server';
```

The certificate is read from a file and passed to **secure** in the following call.

```
CALL secure( xp_read_file('C:\Program Files\SQL Anywhere
12\bin32\rsaserver.id' ) );
```

This example is for illustration only. The certificate can be read directly from a file using the **file=** keyword for the CERTIFICATE clause.

### No matching parameter name example

Placeholders with no matching parameter name are automatically deleted.

For example, the parameter **size** would not be substituted for the placeholder in the following procedure:

```
CREATE PROCEDURE orderitem (size CHAR(18))
URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
TYPE 'SOAP:RPC';
```

In this example, **!sizeXL** is always deleted because it is a placeholder for which there is no matching parameter.

Parameters can be used to replace placeholders within the body of the stored function or stored procedure at the time the function or procedure is called. If placeholders for a particular variable do not exist, the parameter and its value are passed as part of the request. Parameters and values used for substitution in this manner are not passed as part of the request.

## HTTP and SOAP request structures

All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

Parameter values that are not of character or binary data types are converted to a string representation before being added to the request. This process is equivalent to casting the value to a character type. The conversion is done in accordance with the data type formatting option settings at the time the function or procedure is invoked. In particular, the conversion can be affected by such options as precision, scale, and timestamp\_format.

### HTTP request structures

Parameters for type HTTP:GET are URL encoded and placed within the URL. Parameter names are used verbatim as the name for HTTP variables. For example, the following procedure declares two parameters:

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
  URL 'HTTP://localhost/mysevice'
  TYPE 'HTTP:GET';
```

If this procedure is invoked with the two values 123 and 'xyz', then the URL used for the request is equivalent to that shown below:

```
HTTP://localhost/mysevice?a=123&b=xyz
```

If the type is HTTP:POST, the parameters and their values are URL encoded and placed within the body of the request. After the headers, the following text appears in the body of the HTTP request for the two parameter and values:

```
a=123&b=xyz
```

### SOAP request structures

Parameters passed to SOAP requests are bundled as part of the request body, as required by the SOAP specification:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:test>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:test>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### See also

- [“Substitution parameters used for clause values” on page 800](#)

## Logging web client information to the web server

Web service client information, including HTTP requests and transport data, can be logged to the web service client log file, and can be enabled by starting the web server with the `-zoc` server option or by setting the `WebClientLogging` server property using the `sa_server_option` system procedure.

### See also

- [“-zoc dbeng12/dbsrv12 server option”](#) [*SQL Anywhere Server - Database Administration*]
- [“sa\\_server\\_option system procedure”](#) [*SQL Anywhere Server - SQL Reference*]

## Web services references

This section provides information on web service references.

### iAnywhere WSDL compiler utility (**wsdlc**)

The iAnywhere WSDL compiler generates a set of Java proxy classes, C# proxy classes, or SQL SOAP client procedures for SQL Anywhere that you include in your application when given a WSDL source that describes a web service.

#### Syntax

```
wsdlc [options] wsdl-uri
```

#### **wsdl-uri**

This is the specification for the WSDL (Web Services Description Language) source (a URL or file).

#### Options

- **-h** Display help text.
- **-v** Display verbose information.
- **-o *output-directory*** Specify an output directory for generated files.
- **-l *language*** Specify a language for the generated files. This is one of **java**, **cs** (for C#), or **sql**. These options must be specified in lowercase letters.
- **-d** Display debug information that may be helpful when contacting iAnywhere customer support.

#### Java-specific options

- **-p *package*** Specify a package name. This permits you to override the default package name.

#### C#-specific options

- **-n *namespace*** Specify a namespace. This permits you to wrap the generated classes in a namespace of your choosing.

#### SQL-specific options

- **-f *filename*** (Required) Specify the name of the output SQL file to which the SQL statements are written. This operation overwrites any existing file of the same name.
- **-p=*prefix*** Specify a prefix for the generated function or procedure names. The default prefix is the service name followed by a period (for example, "WSDish.").
- **-x** Generate procedure definitions rather than function definitions.

WSDLC does not expand complex parameters representing structures or arrays. Such parameters are commented out to allow the database server to automatically create the given procedure or function

without modification. However, in order for the SOAP operation to work, such parameters must be analyzed and manually composed. The process may require using the SQL Anywhere XMLELEMENT function with the XMLATTRIBUTES parameter to generate complex XML representations of structures.

**Remarks**

The Java or C# classes generated by the WSDL compiler are intended for use with QAnywhere. These classes expose web service operations such as method calls. The classes that are generated are:

- The main service binding class (this class inherits from ianywhere.qanywhere.ws.WSBase in the mobile web services runtime).
- A proxy class for each complex type specified in the WSDL file.

For information about the generated proxy classes, see:

The WSDL compiler supports WSDL 1.1 and SOAP 1.1 over HTTP and HTTPS.

**See also**

- [“Use of the XMLELEMENT function” \[SQL Anywhere Server - SQL Usage\]](#)
- [“QAnywhere .NET API reference for web services” \[QAnywhere\]](#)
- [“QAnywhere Java API reference for web services” \[QAnywhere\]](#)

## Web service error code reference

The HTTP server generates standard web service errors when requests fail. These errors are assigned numbers consistent with protocol standards.

The following are some typical errors that you may encounter:

Number	Name	SOAP fault	Description
301	Moved permanently	Server	The requested page has been permanently moved. The server automatically redirects the request to the new location.
304	Not Modified	Server	The server has decided, based on information in the request, that the requested data has not been modified since the last request and so it does not need to be sent again.
307	Temporary Redirect	Server	The requested page has been moved, but this change may not be permanent. The server automatically redirects the request to the new location.
400	Bad Request	Client.BadRequest	The HTTP request is incomplete or malformed.

Number	Name	SOAP fault	Description
401	Authorization Required	Client.Authorization	Authorization is required to use the service, but a valid user name and password were not supplied.
403	Forbidden	Client.Forbidden	You do not have permission to access the database.
404	Not Found	Client.NotFound	The named database is not running on the server, or the named web service does not exist.
408	Request Timeout	Server.RequestTime-out	The maximum connection idle time was exceeded while receiving the request.
411	HTTP Length Required	Client.LengthRe-quired	The server requires that the client include a Content-Length specification in the request. This typically occurs when uploading data to the server.
413	Entity Too Large	Server	The request exceeds the maximum permitted size.
414	URI Too Large	Server	The length of the URI exceeds the maximum allowed length.
500	Internal Server Error	Server	An internal error occurred. The request could not be processed.
501	Not Implemented	Server	The HTTP request method is not GET, HEAD, or POST.
502	Bad Gateway	Server	The document requested resides on a third-party server and the server received an error from the third-party server.
503	Service Unavail-able	Server	The number of connections exceeds the allowed maximum.

Faults are returned to the client as SOAP faults as defined by the following the SOAP version 1.1 standards when a SOAP service fails:

- When an error in the application handling the request generates a `SQLCODE`, a SOAP Fault is returned with a faultcode of `Client`, possibly with a sub-category, such as `Procedure`. The faultstring element within the SOAP Fault is set to a detailed explanation of the error and a detail element contains the numeric `SQLCODE` value.

- In the event of a transport protocol error, the `faultcode` is set to either `Client` or `Server`, depending on the error, `faultstring` is set to the HTTP transport message, such as `404 Not Found`, and the detail element contains the numeric HTTP error value.
- SOAP Fault messages generated due to application errors that return a `SQLCODE` value are returned with an HTTP status of `200 OK`.

The appropriate HTTP error is returned in a generated HTML document if the client cannot be identified as a SOAP client.

## HTTP web service examples

Several sample implementations of web services are located in the `%SQLANYXSAMP12%\SQLAnywhere\HTTP` directory. For more information about the samples, see `%SQLANYXSAMP12%\SQLAnywhere\HTTP\readme.txt`.

## Tutorial: Working with MIME types in a RAW service

This tutorial illustrates how to create a web server that tests the MIME-type setting specified by a web client application.

### Required software

- SQL Anywhere 12

### Competencies and experience

- Familiarity with XML
- Familiarity with MIME types
- Basic knowledge of SQL Anywhere web services

### Goals

- Create and start a new SQL Anywhere web server database.
- Create a RAW web service.
- Set up a procedure that returns the information contained in HTTP requests.
- Create and start a new SQL Anywhere web client database.
- Send an HTTP:POST request of the `xml/text` MIME-type from the web client to the database server.
- Send an HTTP response in XML format from the database server to the web client.

### See also

- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)



- “CREATE FUNCTION statement [Web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “SQL Anywhere as an HTTP web server” on page 729

## Lesson 1: Setting up a web server to receive RAW requests and send RAW responses

In this lesson, you set up a SQL Anywhere web service server running a web service that tests the MIME-type setting of a web client.

### Set up a database server for receiving RAW requests and sending RAW responses

1. Run the following command to create a SQL Anywhere database:

```
dbinit echo
```

2. Start the network database server using the following command:

```
dbsrv12 -xs http(port=8082) -n echo echo.db
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server in Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=echo"
```

4. Create a new SOAP service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE EchoService
TYPE 'RAW'
USER DBA
AUTHORIZATION OFF
SECURE OFF
AS CALL Echo(:valueAsXML);
```

This statement creates a new SOAP service named **EchoService** that generates a RAW type as output. It calls a stored procedure named **Echo** when a web client sends a request to the service. You create the **Echo** procedure in the next step.

5. Create the **Echo** procedure to handle incoming requests. This procedure returns the body of the request.

Execute the following SQL statements in Interactive SQL:

```
CREATE PROCEDURE Echo( text LONG VARCHAR )
BEGIN
  DECLARE body LONG VARCHAR;
  SET body = isnull( http_variable('text'), http_variable('body') );
  IF body IS NULL THEN
    SELECT 'failed';
  ELSE
```

```
        SELECT body;  
    END IF;  
END;
```

**See also**

- “CREATE SERVICE statement [HTTP web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “CREATE PROCEDURE statement [Web service]” [[SQL Anywhere Server - SQL Reference](#)]

## Lesson 2: Setting up a web client to send RAW requests and receive RAW responses

In this lesson, you set up a MIME-type on a new web client. This lesson assumes that you have set up a web server as instructed in the previous lesson. For more information about setting up a database server to receive the requests from the web client described in this lesson, see “[Lesson 1: Setting up a web server to receive RAW requests and send RAW responses](#)” on page 807.

### Set up a database client for sending RAW requests and receiving RAW responses

1. Run the following command to create a SQL Anywhere database:

```
dbinit echo_client
```

2. Start the personal database client using the following command:

```
dbsrv12 echo_client.db
```

3. Connect to the database in Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=echo_client"
```

4. Create a new stored procedure to send requests to a web service.

Execute the following SQL statement in Interactive SQL:

```
CREATE PROCEDURE setMIME(  
    value LONG VARCHAR,  
    mimeType LONG VARCHAR,  
    urlSpec LONG VARCHAR  
)  
URL '!urlSpec'  
TYPE 'HTTP:POST:!mimeType';
```

In the next lesson, the web client passes the necessary variables to the **setMIME** procedure, which requires three parameters. The **urlSpec** parameter indicates which URL to use to connect to the web service, the **mimeType** indicates which MIME-type to use for the HTTP:POST type, and the **value** parameter represents the body of the request that should be returned by the web service.

**Note**

The server sets the default TYPE to SOAP:RPC when the clause is not specified, which does not support custom MIME-types. For more information about the TYPE clause, see “[CREATE PROCEDURE statement \[Web service\]”](#) [[SQL Anywhere Server - SQL Reference](#)].

## Lesson 3: Sending a RAW request and receiving a RAW response

In this lesson, you call the wrapper procedure created in the previous lesson, which sends a request to the web server you created in lesson one. For more information about setting up a web client to send the requests described in this lesson, see [“Lesson 2: Setting up a web client to send RAW requests and receive RAW responses” on page 808](#).

### Note

This lesson contains several references to **localhost**. Use the IP address of the web server from lesson 1 instead of **localhost** if you are not running the web client on the same computer as the web server.

### Send a request

1. Connect to the client database in Interactive SQL if it is not already open from lesson two.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=echo_client"
```

2. Call the wrapper procedure to send the request and obtain the response.

Execute the following SQL statement in Interactive SQL:

```
CALL setMIME('<hello>this is xml</hello>',
            'text/xml',
            'http://localhost:8082/EchoService'
           );
```

The **http://localhost:8082/EchoService** variable indicates that the database server runs on **localhost** and listens on port 8082. The desired SOAP web service is named **EchoService**.

The following result set is displayed in Interactive SQL:

Attribute	Value
Status	HTTP/1.1 200 OK
Body	<hello>this is xml</hello>
Date	Thu, 04 Feb 2010 13:37:23 GMT
Connection	close
Expires	Thu, 04 Feb 2010 13:37:23 GMT
Content-Type	text/plain; charset=windows-1252
Server	SQLAnywhere/12.0.1.1234

The following is representative of the HTTP packet that is sent to the web server:

```
POST /EchoService HTTP/1.0
Date: Thu, 04 Feb 2010 13:37:23 GMT
Host: localhost
```

```
Accept-Charset: windows-1252, UTF-8, *
User-Agent: SQLAnywhere/12.0.1.1234
Content-Type: text/xml; charset=windows-1252
Content-Length: 49
ASA-Id: 1055532613:echo_client:echo:968000
Connection: close
```

```
valueAsXML=<hello>this is xml</hello>
```

The following is the response from the web server:

```
HTTP/1.1 200 OK
Server: SQLAnywhere/12.0.1.1234
Date: Thu, 04 Feb 2010 13:37:23 GMT
Expires: Thu, 04 Feb 2010 13:37:23 GMT
Content-Type: text/plain; charset=windows-1252
Connection: close
```

```
<hello>this is xml</hello>
```

## Tutorial: Using SQL Anywhere to access a SOAP/DISH service

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit value to Celsius.

### Required software

- SQL Anywhere 12

### Competencies and experience

- Familiarity with SOAP
- Basic knowledge of SQL Anywhere web services

### Goals

- Create and start a new SQL Anywhere web server database.
- Create a SOAP web service.
- Set up a procedure that converts a client-supplied Fahrenheit value to a Celsius value.
- Create and start a new SQL Anywhere web client database.
- Send a SOAP request from the web client to the database server.
- Send a SOAP response from the database server to the web client.

### Suggested background reading

- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE FUNCTION statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

- [“SQL Anywhere as an HTTP web server” on page 729](#)

## Lesson 1: Setting up a web server to receive SOAP requests and send SOAP responses

In this lesson, you set up a new database server and create a SOAP service to handle incoming SOAP requests. The server anticipates SOAP requests that provide a Fahrenheit value that needs to be converted to Celsius.

### Set up a database server for receiving SOAP requests and sending SOAP responses

1. Run the following command to create a SQL Anywhere database:

```
dbinit ftc
```

2. Start the network database server using the following command:

```
dbsrv12 -xs http(port=8082) -n ftc ftc.db
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

3. Connect to the database server in Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc"
```

4. Create a new DISH service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE soap_endpoint
  TYPE 'DISH'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA;
```

This statement creates a new DISH service named **soap\_endpoint** that handles incoming SOAP service requests.

5. Create a new SOAP service to handle Fahrenheit to Celsius conversions.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE FtoCService
  TYPE 'SOAP'
  FORMAT 'XML'
  AUTHORIZATION OFF
  USER DBA
  AS CALL FToCConvertor( :temperature );
```

This statement creates a new SOAP service named **FtoCService** that generates XML-formatted strings as output. It calls a stored procedure named **FToCConvertor** when a web client sends a SOAP request to the service. You create the **FToCConvertor** procedure in the next step.

6. Create the **FToCConverter** procedure to handle incoming SOAP requests. This procedure performs the necessary calculations to convert a client-supplied Fahrenheit value to a Celsius value.

Execute the following SQL statements in Interactive SQL:

```

CREATE PROCEDURE FToCConverter( temperature FLOAT )
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE alias LONG VARCHAR;
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;
header_loop:
    LOOP
        SET hd_key = NEXT_SOAP_HEADER( hd_key );
        IF hd_key IS NULL THEN
            -- no more header entries
            LEAVE header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = SOAP_HEADER( hd_key );
            SET xpath = '/*:' || hd_key || '/*:userName';
            SET namespace = SOAP_HEADER( hd_key, 1, '@namespace' );
            SET mustUnderstand = SOAP_HEADER( hd_key, 1,
'mustUnderstand' );
            BEGIN
                -- parse the XML returned in the SOAP header
                DECLARE crsr CURSOR FOR
                    SELECT * FROM OPENXML( hd_entry, xpath )
                        WITH ( alias LONG VARCHAR '@*:alias',
                            first_name LONG VARCHAR '/*:first/text()',
                            last_name LONG VARCHAR '/*:last/text()' );
                OPEN crsr;
                FETCH crsr INTO alias, first_name, last_name;
                CLOSE crsr;
            END;

            -- build a response header
            -- based on the pieces from the request header
            SET authinfo =
                XMLELEMENT( 'Authentication',
                    XMLATTRIBUTES(
                        namespace as xmlns,
                        alias,
                        mustUnderstand ),
                    XMLELEMENT( 'first', first_name ),
                    XMLELEMENT( 'last', last_name ) );
            CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
        END IF;
    END LOOP header_loop;
    SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;

```

The NEXT\_SOAP\_HEADER function is used in a LOOP structure to iterate through all the header names in a SOAP request, and exits the loop when the NEXT\_SOAP\_HEADER function returns NULL.

**Note**

This function does not necessarily iterate through the headers in the order that they appear in the SOAP request.

The SOAP\_HEADER function returns the header value or NULL when the header name does not exist. The **FToCConvertor** procedure searches for a header named **Authentication** and extracts the header structure, including the **@namespace** and **mustUnderstand** attributes. The **@namespace** header attribute is a special SQL Anywhere attribute used to access the namespace (**xmlns**) of the given header entry.

The following is an XML string representation of a possible **Authentication** header structure, where the **@namespace** attribute has a value of "SecretAgent", and **mustUnderstand** has a value of 1:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

The OPENXML system procedure in the SELECT statement parses the XML header using the XPath string **"/\*:Authentication/\*:userName"** to extract the **alias** attribute value and the contents of the **first** and **last** tags. The result set is processed using a cursor to fetch the three column values.

At this point, you have all the information of interest that was passed to the web service. You have the temperature in Fahrenheit degrees and you have some additional attributes that were passed to the web service in a SOAP header. You could look up the name and alias that were provided to see if the person is authorized to use the web service. However, this exercise is not shown in the example.

The SET statement is used to build a SOAP response in XML format to send to the client. The following is an XML string representation of a possible SOAP response. It is based on the above **Authentication** header structure example.

```
<Authentication xmlns="SecretAgent" alias="99" mustUnderstand="1">
  <first>Susan</first>
  <last>Hilton</last>
</Authentication>
```

The SA\_SET\_SOAP\_HEADER system procedure is used to set the SOAP response header that will be sent to the client.

The final SELECT statement is used to convert the supplied Fahrenheit value to a Celsius value. This information is relayed back to the client.

At this point, you now have a running SQL Anywhere web server that provides a service for converting temperatures from degrees Fahrenheit to degrees Celsius. This service processes a SOAP header from the client and sends a SOAP response back to the client. In the next section, you develop an example of a client that can send SOAP requests to the web server and receive SOAP responses from the web server.

**See also**

- “CREATE SERVICE statement [SOAP web service]” [*SQL Anywhere Server - SQL Reference*]
- “CREATE PROCEDURE statement [Web service]” [*SQL Anywhere Server - SQL Reference*]
- “SOAP\_HEADER function [SOAP]” [*SQL Anywhere Server - SQL Reference*]
- “NEXT\_SOAP\_HEADER function [SOAP]” [*SQL Anywhere Server - SQL Reference*]
- “SOAP namespace URI requirement” on page 780
- “openxml system procedure” [*SQL Anywhere Server - SQL Reference*]

## Lesson 2: Setting up a web client to send SOAP requests and receive SOAP responses

In this lesson, you set up a web client that sends SOAP requests and receives SOAP responses. This lesson assumes that you have set up a web server as instructed in the previous lesson. For more information about setting up a database server to process SOAP requests from the client described in this lesson, see “Lesson 1: Setting up a web server to receive SOAP requests and send SOAP responses” on page 811.

**Note**

This lesson contains several references to **localhost**. Use the IP address of the web server from lesson 1 instead of **localhost** if you are not running the web client on the same computer as the web server.

### Set up a client for sending SOAP requests and receiving SOAP responses

1. Run the following command to create a SQL Anywhere database:

```
dbinit ftc_client
```

2. Start the personal database client using the following command:

```
dbsrv12 ftc_client.db
```

3. Connect to the database in Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc_client"
```

4. Create a new stored procedure to send SOAP requests to a DISH service.

Execute the following SQL statement in Interactive SQL:

```
CREATE PROCEDURE FtoC( temperature FLOAT,  
    INOUT inoutheader LONG VARCHAR,  
    IN inheader LONG VARCHAR )  
URL 'http://localhost:8082/soap_endpoint'  
SET 'SOAP(OP=FtoCService)'  
TYPE 'SOAP:DOC'  
SOAPHEADER '!inoutheader!inheader';
```

The **http://localhost:8082/soap\_endpoint** string in the URL clause indicates that the web server runs on **localhost** and listens on port 8082. The desired DISH web service is named **soap\_endpoint**, which serves as a SOAP endpoint.

The SET clause specifies the name of the SOAP operation or service **FtoCService** that is to be called.



The default format used when making a web service request is 'SOAP:RPC'. The format chosen in this example is 'SOAP:DOC', which is similar to 'SOAP:RPC' but allows for a richer set of datatypes. SOAP requests are always sent as XML documents. The mechanism for sending SOAP requests is 'HTTP:POST'.

The substitution variables in a SQL Anywhere client procedure (**inoutheader**, **inheader**) must be alpha-numeric. If the web service client is declared as a function, all its parameters are IN mode only (they cannot be assigned by the called function). Therefore, OPENXML or other string functions would have to be used to extract the SOAP response header information.

5. Create a wrapper procedure that builds two special SOAP request header entries, passes them to the **FtoC** procedure, and processes server responses.

Execute the following SQL statements in Interactive SQL:

```
CREATE PROCEDURE FahrenheitToCelsius( temperature FLOAT )
BEGIN
  DECLARE io_header LONG VARCHAR;
  DECLARE in_header LONG VARCHAR;
  DECLARE result LONG VARCHAR;
  DECLARE err INTEGER;
  DECLARE crsr CURSOR FOR
    CALL FtoC( temperature, io_header, in_header );
  SET io_header =
    '<Authentication xmlns="SecretAgent" ' ||
    'mustUnderstand="1">' ||
    '<userName alias="99">' ||
    '<first>Susan</first><last>Hilton</last>' ||
    '</userName>' ||
    '</Authentication>';
  SET in_header =
    '<Session xmlns="SomeSession">' ||
    '123456789' ||
    '</Session>';

  MESSAGE 'send, soapheader=' || io_header || in_header;
  OPEN crsr;
  FETCH crsr INTO result, err;
  CLOSE crsr;
  MESSAGE 'receive, soapheader=' || io_header;
  SELECT temperature, Celsius
    FROM OPENXML(result, '//tns:answer', 1, result)
    WITH ("Celsius" FLOAT 'text()');
END;
```

The first SET statement creates the XML representation of a SOAP header entry to inform the web server of user credentials:

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

The second SET statement creates the XML representation of a SOAP header entry to track the client session ID:

```
<Session xmlns="SomeSession">123456789</Session>
```

6. The OPEN statement causes the FtoC procedure to be called which sends a SOAP request to the web server and then processes the response from the web server. The response includes a header which is returned in **inoutheader**.

In the next section, you send a SOAP request to the web server and examine the SOAP response.

### See also

- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Lesson 3: Sending a SOAP request and receiving a SOAP response

In this lesson, you call the wrapper procedure created in the previous lesson, which sends a SOAP request to the web server that you created in lesson one. For more information about setting up a web client to send the SOAP requests described in this lesson, see [“Lesson 2: Setting up a web client to send SOAP requests and receive SOAP responses”](#) on page 814.

### Send a SOAP request

1. Connect to the client database in Interactive SQL if it is not already open from lesson two.

```
dbisql -c "UID=DBA;PWD=sql;SERVER=ftc_client"
```

2. Enable logging of SOAP requests and responses.

Execute the following SQL statements in Interactive SQL:

```
CALL sa_server_option('WebClientLogFile', 'soap.txt');  
CALL sa_server_option('WebClientLogging', 'ON');
```

These calls allow you to examine the content of the SOAP request and response. The requests and responses are logged to a file called *soap.txt*.

3. Call the wrapper procedure to send a SOAP request and receive the SOAP response.

Execute the following SQL statement in Interactive SQL:

```
CALL FahrenheitToCelsius(212);
```

This call passes a Fahrenheit value of 212 to the **FahrenheitToCelsius** procedure, which passes the value along with two customized SOAP headers to the **FToC** procedure. Both client-side procedures are created in the previous lesson.

The **FToC** web service procedure sends the Fahrenheit value and the SOAP headers to the web server. The SOAP request contains the following.

```
<SOAP-ENV:Envelope  
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:m="http://localhost:8082">  
  <SOAP-ENV:Header>  
    <Authentication xmlns="SecretAgent" mustUnderstand="1">  
      <userName alias="99">
```

```

        <first>Susan</first>
        <last>Hilton</last>
    </userName>
</Authentication>
<Session xmlns="SomeSession">
    123456789
</Session>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <m:FtoCService>
        <m:temperature>212</m:temperature>
    </m:FtoCService>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The FtoC procedure then receives the response from the web server which includes a result set based on the Fahrenheit value. The SOAP response contains the following.

```

<SOAP-ENV:Envelope
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:tns='http://localhost:8082'>
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" alias="99" mustUnderstand="1">
      <first>Susan</first>
      <last>Hilton</last>
    </Authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <tns:FtoCServiceResponse>
      <tns:FtoCServiceResult xsi:type='xsd:string'>
        &lt;tns:rowset xmlns:tns="http://localhost:8082/
ftc" >
          &lt;tns:row >
            &lt;tns:answer > 20
          &lt;/tns:row >
          &lt;tns:rowset >
        </tns:FtoCServiceResult>
      <tns:sqlcode>0</tns:sqlcode>
    </tns:FtoCServiceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The content of **<SOAP-ENV:Header>** is returned in **inoutheader**.

If you examine the SOAP response, you can see that the result set was encoded in the response by the **FToCService** web service. The result set is decoded and returned to the **FahrenheitToCelsius** procedure. The result set looks like the following when a Fahrenheit value of 212 is passed to the web server:

```

<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
    <tns:answer>100
  </tns:row>
</tns:rowset>

```

The SELECT statement in the **FahrenheitToCelsius** procedure uses the OPENXML function to parse the SOAP response, extracting the Celsius value defined by the **tns:answer** structure.

The following result set is generated in Interactive SQL:

temperature	Celsius
212	100

**See also**

- [“sa\\_server\\_option system procedure” \[SQL Anywhere Server - SQL Reference\]](#)

## Tutorial: Using JAX-WS to access a SOAP/DISH web service

This tutorial illustrates how to create a Java API for XML Web Services (JAX-WS) client application to access SOAP/DISH services on a SQL Anywhere web server.

**Required software**

- SQL Anywhere 12
- Java SE 6
- JAX-WS 2.2 (or a later version) for Windows (installation instructions are provided in the tutorial)

**Competencies and experience**

- Familiarity with SOAP
- Familiarity with Java or JAX-WS
- Basic knowledge of SQL Anywhere web services

**Goals**

- Create and start a new SQL Anywhere web server database.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Set up JAX-WS on the client computer and import a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

**Suggested background reading**

- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

- “CREATE FUNCTION statement [Web service]” [[SQL Anywhere Server - SQL Reference](#)]
- “SQL Anywhere as an HTTP web server” on page 729

## Lesson 1: Setting up a web server to receive SOAP requests and send SOAP responses

In this lesson, you set up a SQL Anywhere web server running SOAP and DISH web services that handles JAX-WS client application requests.

### Set up a database server for receiving RAW requests and sending RAW responses

1. Start the SQL Anywhere **demo** database using the following command:

```
dbsrv12 -xs http(port=8082) "%SQLANYAMP12%\demo.db"
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

2. Connect to the database server with Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
```

3. Create a stored procedure that lists **Employees** table columns.

Execute the following SQL statements in Interactive SQL:

```
CREATE PROCEDURE ListEmployees()
RESULT (
    EmployeeID          INTEGER,
    Surname              CHAR(20),
    GivenName           CHAR(20),
    StartDate           DATE,
    TerminationDate     DATE )
BEGIN
    SELECT EmployeeID, Surname, GivenName, StartDate, TerminationDate
    FROM Employees;
END;
```

These statements create a new procedure named **ListEmployees** that defines the structure of the result set output, and selects certain columns from the Employees table.

4. Create a new SOAP service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "WS/EmployeeList"
TYPE 'SOAP'
FORMAT 'CONCRETE' EXPLICIT ON
DATATYPE ON
AUTHORIZATION OFF
SECURE OFF
USER DBA
AS CALL ListEmployees();
```

This statement creates a new SOAP web service named **WS/EmployeeList** that generates a SOAP type as output. It calls the **ListEmployees** procedure when a web client sends a request to the service.

SOAP web services accessed from JAX-WS should be declared with the `FORMAT 'CONCRETE'` clause. The `EXPLICIT ON` clause indicates that the corresponding DISH service should generate XML Schema that describes an explicit dataset object based on the result set of the **ListEmployees** procedure. The `EXPLICIT` clause only affects the generated WSDL document. For more information about the `EXPLICIT` clause, see “[CREATE PROCEDURE statement \[Web service\]](#)” [*SQL Anywhere Server - SQL Reference*].

`DATATYPE ON` indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated. For more information about the `DATATYPE` clause, see “[CREATE PROCEDURE statement \[Web service\]](#)” [*SQL Anywhere Server - SQL Reference*].

5. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "WSDish"  
  TYPE 'DISH'  
  FORMAT 'CONCRETE'  
  GROUP "WS"  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA;
```

DISH web services accessed from JAX-WS should be declared with the `FORMAT 'CONCRETE'` clause. The `GROUP` clause identifies the SOAP services that should be handled by the DISH service. The `EmployeeList` service created in the previous step is part of the `GROUP WS` because it is declared as **WS/EmployeeList**.

6. Verify that the DISH web service is functional by accessing it through a web browser.

Open your web browser and go to <http://localhost:8082/demo/WSDish>.

The DISH service automatically generates a WSDL document that appears in the browser window. Examine the **EmployeeListDataset** object, which looks similar to the following output:

```
<s:complexType name="EmployeeListDataset">  
  <s:sequence>  
    <s:element name="rowset">  
      <s:complexType>  
        <s:sequence>  
          <s:element name="row" minOccurs="0" maxOccurs="unbounded">  
            <s:complexType>  
              <s:sequence>  
                <s:element minOccurs="0" maxOccurs="1" name="EmployeeID"  
                  nillable="true" type="s:int" />  
                <s:element minOccurs="0" maxOccurs="1" name="Surname"  
                  nillable="true" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="GivenName"  
                  nillable="true" type="s:string" />  
                <s:element minOccurs="0" maxOccurs="1" name="StartDate"
```

```

        nillable="true" type="s:date" />
    <s:element minOccurs="0" maxOccurs="1" name="TerminationDate"
        nillable="true" type="s:date" />
    </s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
</s:sequence>
</s:complexType>

```

**EmployeeListDataset** is the explicit object generated by the FORMAT 'CONCRETE' and EXPLICIT ON clauses in the **EmployeeList** SOAP service. In a later lesson, the wsimport application uses this information to generate a SOAP 1.1 client interface for this service.

7. (optional) Restore the sample database (*demo.db*) to its original state. See [“Recreate the sample database \(demo.db\)” \[SQL Anywhere 12 - Introduction\]](#).

#### See also

- [“CREATE SERVICE statement \[SOAP web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)

## Lesson 2: Setting up JAX-WS on the web client

In this lesson, you set up JAX-WS on the web client to gather database information from the web server. It assumes that you have already set up the Java SE 6 JDK in the *C:\Program Files\Java\JDK1.6.0* directory. Note that, at the time of writing, recent versions of the JDK include JAX-WS 2.1.6. To determine if JAX-WS is present, check for a **wsimport** command in the *C:\Program Files\Java\JDK1.6.0\bin* directory. If you wish to install a more recent version such as JAX-WS 2.2, then use the following procedure.

### Set up the latest version of JAX-WS on a web client

1. Download the latest version of JAX-WS. Go to <http://jax-ws.java.net/> to download the latest version.
2. Install JAX-WS.

Follow the online instructions to install JAX-WS. This tutorial assumes that all files are extracted to the *C:\Program Files\Java* directory.

3. Add the *C:\Program Files\Java\jaxws-ri\bin* and *C:\Program Files\Java\JDK1.6.0\bin* directories to your **PATH** environment variable in the order listed.

The code used in this tutorial relies on the JDK and JAX-WS binaries. You can write and compile the code from any directory by adding paths to these locations to your **PATH** environment variable.

4. Set your **CLASSPATH** environment variable using the following command:

```

SET classpath=.;C:\Program Files\Java\jaxws-ri\lib\jaxb-api.jar;
C:\Program Files\Java\jaxws-ri\lib\jaxws-rt.jar

```

5. Set the **JAXWS\_HOME** environment variable using the following command:

```
SET JAXWS_HOME=C:\Program Files\Java\jaxws-ri
```

6. Override endorsed standards provided by the Java Runtime Environment (JRE). To override the standard run-time environment, create an *endorsed* directory under the Java Runtime Environment (JRE) *lib* directory. This is usually *C:\Program Files\Java\JDK1.6.0\jre\lib*. Do not place the *endorsed* directory under the JDK *lib* directory.

Copy the *jaxws-ri\lib\jaxws-api.jar* and *jaxws-ri\lib\jaxb-api.jar* files to the *C:\Program Files\Java\JDK1.6.0\jre\lib\endorsed* directory.

**Note**

JAX-WS 2.2 provides additional functionality to packaged objects that are not provided in current versions of the JDK. You must override the endorsed standards to make use of this functionality and prevent runtime errors. For more information about the Java Endorsed Standards Override Mechanism, see <http://docs.oracle.com/javase/6/docs/technotes/guides/standards/>.

## Lesson 3: Creating a Java application to communicate with the web server

In this lesson, you import the WSDL document generated from the DISH service and create a Java application to access table data based on the schema defined in the WSDL document.

### Import a WSDL document from the database server

1. At a command prompt, create a new working directory for your Java code and generated files. Change to this new directory.
2. Generate the interface that calls the DISH web service and imports the WSDL document using the following command:

```
wsimport -keep -Xendorsed "http://localhost:8082/demo/WSDish"
```

The `wsimport` application retrieves the WSDL document from the given URL. It generates *.java* files to create an interface for it, then compiles them into *.class* files.

The **keep** option indicates that the *.java* files should not be deleted after generating the class files. These files allow you to understand the generated source code.

The **Xendorsed** option enables use of the endorsed standards overloaded in a previous lesson.

The `wsimport` application creates a new subdirectory structure named *localhost\_8082\demo\ws* in your current working directory. The following is a list of directory contents:

- *EmployeeList.class*
- *EmployeeList.java*



- *EmployeeListDataset\$Rowset\$Row.class*
- *EmployeeListDataset\$Rowset.class*
- *EmployeeListDataset.class*
- *EmployeeListDataset.java*
- *EmployeeListResponse.class*
- *EmployeeListResponse.java*
- *FaultMessage.class*
- *FaultMessage.java*
- *ObjectFactory.class*
- *ObjectFactory.java*
- *package-info.class*
- *package-info.java*
- *WSDish.class*
- *WSDish.java*
- *WSDishSoapPort.class*
- *WSDishSoapPort.java*

### Access database information from the server

1. Write a Java application that accesses table data from the database server based on the dataset object schema defined in the generated API.

Save the following Java source code as *SASoapDemo.java* in the current working directory. Note that your current working directory must be the directory containing the *localhost* subdirectory.

```
// SASoapDemo.java illustrates a web service client that
// calls the WSDish service and prints out the data.

import java.util.*;
import javax.xml.ws.*;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import javax.xml.datatype.*;
import localhost._8082.demo.ws.*;

public class SASoapDemo
{
    public static void main( String[] args )
    {
        try {
```

```

WSDish service = new WSDish();

Holder<EmployeeListDataset> response =
    new Holder<EmployeeListDataset>();
Holder<Integer> sqlcode = new Holder<Integer>();

WSDishSoapPort port = service.getWSDishSoap();

// This is the SOAP service call to EmployeeList
port.employeeList( response, sqlcode );

EmployeeListDataset result = response.value;
EmployeeListDataset.Rowset rowset = result.getRowset();

List<EmployeeListDataset.Rowset.Row> rows = rowset.getRow();

String fieldType;
String fieldName;
String fieldValue;
Integer fieldInt;
XMLGregorianCalendar fieldDate;

for ( int i = 0; i < rows.size(); i++ ) {
    EmployeeListDataset.Rowset.Row row = rows.get( i );

    fieldType = row.getEmployeeID().getDeclaredType().getSimpleName();
    fieldName = row.getEmployeeID().getName().getLocalPart();
    fieldInt = row.getEmployeeID().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
        "=" + fieldInt );

    fieldType = row.getSurname().getDeclaredType().getSimpleName();
    fieldName = row.getSurname().getName().getLocalPart();
    fieldValue = row.getSurname().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
        "=" + fieldValue );

    fieldType = row.getGivenName().getDeclaredType().getSimpleName();
    fieldName = row.getGivenName().getName().getLocalPart();
    fieldValue = row.getGivenName().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
        "=" + fieldValue );

    fieldType = row.getStartDate().getDeclaredType().getSimpleName();
    fieldName = row.getStartDate().getName().getLocalPart();
    fieldDate = row.getStartDate().getValue();
    System.out.println( "(" + fieldType + ")" + fieldName +
        "=" + fieldDate );

    if ( row.getTerminationDate() == null ) {
        fieldType = "unknown";
        fieldName = "TerminationDate";
        fieldDate = null;
    } else {
        fieldType =
            row.getTerminationDate().getDeclaredType().getSimpleName();
        fieldName = row.getTerminationDate().getName().getLocalPart();
        fieldDate = row.getTerminationDate().getValue();
    }
    System.out.println( "(" + fieldType + ")" + fieldName +
        "=" + fieldDate );
    System.out.println();
}
}

```

```

        catch (Exception x) {
            x.printStackTrace();
        }
    }
}

```

This application prints all server-provided column data to the standard system output.

**Note**

This application assumes that your SQL Anywhere web server is listening on port 8082, as instructed in lesson one. Replace the **8082** portion of the **import localhost.\_8082.demo.ws.\*** code line with the port number you specified when you started the SQL Anywhere web server.

For more information about the Java methods used in this application, see the `javax.xml.bind.JAXBElement` class documentation at <http://docs.oracle.com/javase/6/api/>.

2. Compile your Java application using the following command:

```
javac SASoapDemo.java
```

3. Execute the application using the following command:

```
java SASoapDemo
```

4. The application sends its request to the web server. It receives an XML result set response that consists of an **EmployeeListResult** with a rowset containing several row entries. The SQLCODE result from executing the query is included in the response.

The following is an example of the generated output:

```

(Integer)EmployeeID=102
(String)Surname=Whitney
(String)GivenName=Fran
(XMLGregorianCalendar)StartDate=1984-08-28
(unknown)TerminationDate=null

(Integer)EmployeeID=105
(String)Surname=Cobb
(String)GivenName=Matthew
(XMLGregorianCalendar)StartDate=1985-01-01
(unknown)TerminationDate=null
.
.
(Integer)EmployeeID=1740
(String)Surname=Nielsen
(String)GivenName=Robert
(XMLGregorianCalendar)StartDate=1994-06-24
(unknown)TerminationDate=null

(Integer)EmployeeID=1751
(String)Surname=Ahmed
(String)GivenName=Alex
(XMLGregorianCalendar)StartDate=1994-07-12
(XMLGregorianCalendar)TerminationDate=2008-04-18

```

The **TerminationDate** column is only sent when its value is not NULL. The Java application is designed to detect when the **TerminationDate** column is not present. For this example, the last row in the Employees table was altered such that a non-NULL termination date was set.

The following is an example of a SOAP response from the web server:

```
<tns:EmployeeListResponse>
  <tns:EmployeeListResult xsi:type='tns:EmployeeListDataset'>
    <tns:rowset>
      <tns:row> ... </tns:row>
      .
      .
      .
    <tns:row>
      <tns:EmployeeID xsi:type="xsd:int">1751</tns:EmployeeID>
      <tns:Surname xsi:type="xsd:string">Ahmed</tns:Surname>
      <tns:GivenName xsi:type="xsd:string">Alex</tns:GivenName>
      <tns:StartDate xsi:type="xsd:dateTime">1994-07-12</tns:StartDate>
      <tns:TerminationDate xsi:type="xsd:dateTime">2010-03-22</
tns:TerminationDate>
    </tns:row>
  </tns:rowset>
</tns:EmployeeListResult>
<tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeListResponse>
```

Column names and data types are included in each rowset.

**Note**

You can observe the response shown above through the use of proxy software that logs the XML message traffic. The proxy inserts itself between your client application and the web server.

## Tutorial: Using Visual C# to access a SOAP/DISH web service

This tutorial illustrates how to create a Visual C# client application to access SOAP/DISH services on a SQL Anywhere web server.

### Required software

- SQL Anywhere 12
- Visual Studio

### Competencies and experience

- Familiarity with SOAP
- Familiarity with .NET framework
- Basic knowledge of SQL Anywhere web services

## Goals

- Create and start a new SQL Anywhere web server database.
- Create a SOAP web service.
- Set up a procedure that returns the information contained in a SOAP request.
- Create a DISH web service that provides WSDL documents and acts as a proxy.
- Set up Visual C# on the client computer and import a WSDL document from the web server.
- Create a Java client application to retrieve information from the SOAP service using the WSDL document information.

## Suggested background reading

- [“CREATE PROCEDURE statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE FUNCTION statement \[Web service\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“SQL Anywhere as an HTTP web server” on page 729](#)

## Lesson 1: Setting up a web server to receive SOAP requests and send SOAP responses

In this lesson, you set up a SQL Anywhere web server running SOAP and DISH web services that handles Visual C# client application requests.

### Set up a database server for receiving RAW requests and sending RAW responses

1. Start the SQL Anywhere **demo** database using the following command:

```
dbsrv12 -xs http(port=8082) "%SQLANYSAMP12%\demo.db"
```

This command indicates that the HTTP web server should listen on port 8082 for requests. Use a different port number if 8082 is disallowed on your network.

2. Connect to the database server in Interactive SQL using the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=demo"
```

3. Create a new SOAP service to accept incoming requests.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "SASoapTest/EmployeeList"  
TYPE 'SOAP'  
DATATYPE ON  
AUTHORIZATION OFF  
SECURE OFF  
USER DBA  
AS SELECT * FROM Employees;
```

This statement creates a new SOAP web service named **SASoapTest/EmployeeList** that generates a SOAP type as output. It selects all columns from the **Employees** table and returns the result set to the client.

DATATYPE ON indicates that explicit data type information is generated in the XML result set response and the input parameters. This option does not affect the WSDL document that is generated. For more information about the DATATYPE clause, see “[CREATE PROCEDURE statement \[Web service\]](#)” [*SQL Anywhere Server - SQL Reference*].

4. Create a new DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

Execute the following SQL statement in Interactive SQL:

```
CREATE SERVICE "SASoapTest_DNET"  
TYPE 'DISH'  
GROUP "SASoapTest"  
FORMAT 'DNET'  
AUTHORIZATION OFF  
SECURE OFF  
USER DBA;
```

DISH web services accessed from .NET should be declared with the FORMAT 'DNET' clause. The GROUP clause identifies the SOAP services that should be handled by the DISH service. The **EmployeeList** service created in the previous step is part of the GROUP 'SASoapTest' because it is declared as **SASoapTest/EmployeeList**.

5. Verify that the DISH web service is functional by accessing it through a web browser.

Open your web browser and go to [http://localhost:8082/demo/SASoapTest\\_DNET](http://localhost:8082/demo/SASoapTest_DNET).

### See also

- “[CREATE SERVICE statement \[SOAP web service\]](#)” [*SQL Anywhere Server - SQL Reference*]
- “[Creating DISH services](#)” on page 738

## Lesson 2: Creating a Visual C# application to communicate with the web server

Note that this example uses functions from the .NET Framework 2.0.

### Create SOAP and DISH services

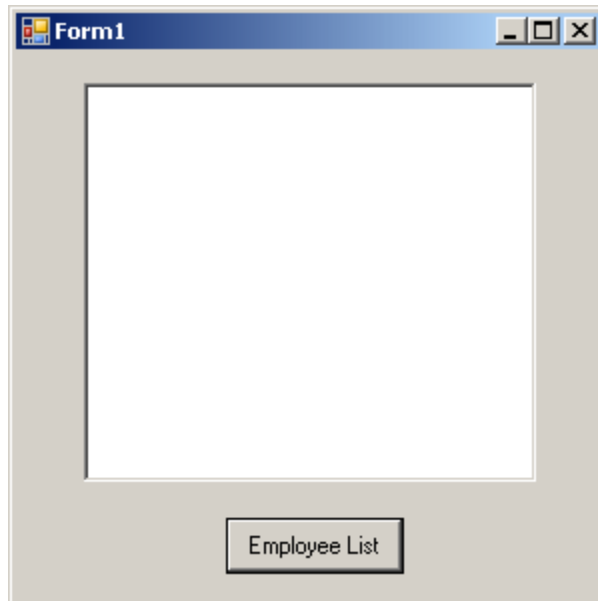
1. Start Visual Studio.
2. Create a new Visual C# Windows form application project.

An empty form appears.

3. Add a web reference to the project.
  - a. Click **Project** » **Add Service Reference**.

- b. In the Add Service Reference window, click **Advanced**.
  - c. In the Service Reference Settings window, click **Add Web Reference**.
  - d. In the Add Web Reference window, type **http://localhost:8082/demo/SASoapTest\_DNET** in the **URL** field.
  - e. Click **Go**.  
Visual Studio lists the EmployeeList method available from the SASoapTest\_DNET service.
  - f. Click **Add Reference**.  
Visual Studio adds the **localhost** web reference to the project. You can select it from the **Solution Explorer** pane.
4. Populate the empty form with the desired objects for web client application.

From the **Toolbox** pane, drag ListBox and Button objects onto the form and update the text attributes so that the form looks similar to the following diagram:



5. Write a procedure that accesses the web reference and uses the available methods.

Double-click **Employee List** and add the following code for the button click event:

```
int sqlCode;

listBox1.Items.Clear();

localhost.SASoapTest_DNET proxy = new localhost.SASoapTest_DNET();

DataSet results = proxy.EmployeeList(out sqlCode);
DataTableReader dr = results.CreateDataReader();
while (dr.Read())
{
```

```

for (int i = 0; i < dr.FieldCount; i++)
{
    string columnName = "(" + dr.GetDataTypeName(i)
                        + ")"
                        + dr.GetName(i);
    if (dr.IsDBNull(i))
    {
        listBox1.Items.Add(columnName + "(null)");
    }
    else {
        System.TypeCode typeCode =
            System.Type.GetTypeCode(dr.GetFieldType(i));
        switch (typeCode)
        {
            case System.TypeCode.Int32:
                Int32 intValue = dr.GetInt32(i);
                listBox1.Items.Add(columnName + "="
                    + intValue);
                break;
            case System.TypeCode.Decimal:
                Decimal decValue = dr.GetDecimal(i);
                listBox1.Items.Add(columnName + "="
                    + decValue.ToString("c"));
                break;
            case System.TypeCode.String:
                string stringValue = dr.GetString(i);
                listBox1.Items.Add(columnName + "="
                    + stringValue);
                break;
            case System.TypeCode.DateTime:
                DateTime dateValue = dr.GetDateTime(i);
                listBox1.Items.Add(columnName + "="
                    + dateValue);
                break;
            case System.TypeCode.Boolean:
                Boolean boolValue = dr.GetBoolean(i);
                listBox1.Items.Add(columnName + "="
                    + boolValue);
                break;
            case System.TypeCode.DBNull:
                listBox1.Items.Add(columnName
                    + "(null)");
                break;
            default:
                listBox1.Items.Add(columnName
                    + "(unsupported)");
                break;
        }
    }
}
listBox1.Items.Add("");
}
dr.Close();

```

6. Run the application.

Click **Debug » Start Debugging**.

7. Communicate with the web database server.

Click **Employee List**.



The **ListBox** object displays the **EmployeeList** result set as (type)name=value pairs. The following output illustrates how an entry appears in the **ListBox** object:

```
(Int32)EmployeeID=102
(Int32)ManagerID=501
(String)Surname=Whitney
(String)GivenName=Fran
(Int32)DepartmentID=100
(String)Street=9 East Washington Street
(String)City=Cornwall
(String)State=New York
(String)Country=USA
(String)PostalCode=02192
(String)Phone=6175553985
(String)Status=A
(String)SocialSecurityNumber=017349033
(String)Salary=$45,700.00
(DateTime)StartDate=28/08/1984 0:00:00 AM
(DateTime)TerminationDate=(null)
(DateTime)BirthDate=05/06/1958 0:00:00 AM
(Boolean)BenefitHealthInsurance=True
(Boolean)BenefitLifeInsurance=True
(Boolean)BenefitDayCare=False
(String)Sex=F
```

The Salary amount is converted to the client's currency format.

Values that contain null are returned as DBNull. Values that contain a date with no time are assigned a time of 00:00:00 or midnight.

The XML response from the web server includes a formatted result set. All column data is converted to a string representation of the data. The following result set illustrates how result sets are formatted when they are sent to the client:

```
<row>
  <EmployeeID>102</EmployeeID>
  <ManagerID>501</ManagerID>
  <Surname>Whitney</Surname>
  <GivenName>Fran</GivenName>
  <DepartmentID>100</DepartmentID>
  <Street>9 East Washington Street</Street>
  <City>Cornwall</City>
  <State>NY</State>
  <Country>USA</Country>
  <PostalCode>02192</PostalCode>
  <Phone>6175553985</Phone>
  <Status>A</Status>
  <SocialSecurityNumber>017349033</SocialSecurityNumber>
  <Salary>45700.000</Salary>
  <StartDate>1984-08-28-05:00</StartDate>
  <TerminationDate xsi:nil="true" />
  <BirthDate>1958-06-05-05:00</BirthDate>
  <BenefitHealthInsurance>1</BenefitHealthInsurance>
  <BenefitLifeInsurance>1</BenefitLifeInsurance>
  <BenefitDayCare>0</BenefitDayCare>
  <Sex>F</Sex>
</row>
```

Columns containing date or time information include the offset from UTC of the web server. In the above result set, the offset is -05:00 which is 5 hours to the west of UTC (North American Eastern Standard Time).

Columns containing only the date are formatted as `yyyy-mm-dd-HH:MM` or `yyyy-mm-dd+HH:MM`. A zone offset (`-HH:MM` or `+HH:MM`) is suffixed to the string.

Columns containing only the time are formatted as `hh:mm:ss.nnn-HH:MM` or `hh:mm:ss.nnn+HH:MM`. A zone offset (`-HH:MM` or `+HH:MM`) is suffixed to the string.

Columns containing both date and time are formatted as `yyyy-mm-ddThh:mm:ss.nnn-HH:MM` or `yyyy-mm-ddThh:mm:ss.nnn+HH:MM`. Note that the date is separated from the time using the letter 'T'. A zone offset (`-HH:MM` or `+HH:MM`) is suffixed to the string.

The DATATYPE ON clause is specified to generate data type information in the XML result set response. A fragment of the response from the web server is shown below. Note that the type information matches the data type of the database columns.

```
<xsd:element name='EmployeeID' minOccurs='0' type='xsd:int' />
<xsd:element name='ManagerID' minOccurs='0' type='xsd:int' />
<xsd:element name='Surname' minOccurs='0' type='xsd:string' />
<xsd:element name='GivenName' minOccurs='0' type='xsd:string' />
<xsd:element name='DepartmentID' minOccurs='0' type='xsd:int' />
<xsd:element name='Street' minOccurs='0' type='xsd:string' />
<xsd:element name='City' minOccurs='0' type='xsd:string' />
<xsd:element name='State' minOccurs='0' type='xsd:string' />
<xsd:element name='Country' minOccurs='0' type='xsd:string' />
<xsd:element name='PostalCode' minOccurs='0' type='xsd:string' />
<xsd:element name='Phone' minOccurs='0' type='xsd:string' />
<xsd:element name='Status' minOccurs='0' type='xsd:string' />
<xsd:element name='SocialSecurityNumber' minOccurs='0' type='xsd:string' />
<xsd:element name='Salary' minOccurs='0' type='xsd:decimal' />
<xsd:element name='StartDate' minOccurs='0' type='xsd:date' />
<xsd:element name='TerminationDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BirthDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BenefitHealthInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitLifeInsurance' minOccurs='0' type='xsd:boolean' />
<xsd:element name='BenefitDayCare' minOccurs='0' type='xsd:boolean' />
<xsd:element name='Sex' minOccurs='0' type='xsd:string' />
```

## Sample: Handling SOAP headers, parameters, and responses

This sample illustrates different ways of exchanging SOAP requests between a web server and a SOAP client.

The following server-side code illustrates how a web server can process SOAP requests containing parameters, and SOAP headers. The example implements an **addItem** SOAP operation that takes two parameters: **amount** of type `int` and **item** of type `string`. The **sp\_addItems** procedure also processes an Authentication SOAP header extracting the first and last name of the user. The values are used to populate a SOAP response Validation header via the `sa_set_soap_header` system procedure. The response is a result of three columns: **quantity**, **item** and **status** with types `INT`, `LONG VARCHAR` and `LONG VARCHAR` respectively.

```
// create the SOAP service
call sa_make_object( 'service', 'addItem' );
alter SERVICE "addItem"
```

```

TYPE 'SOAP'
format 'concrete'
AUTHORIZATION OFF
USER dba
AS call sp_addItems( :amount, :item );

// create SOAP endpoint for related services
call sa_make_object( 'service', 'store' );
alter SERVICE "store"
TYPE 'DISH'
AUTHORIZATION OFF
USER dba;

// create the procedure that will process the SOAP requests for the addItems
service
create or replace procedure sp_addItems( "count" int, item long varchar )
result( quantity int, item long varchar, status long varchar )
begin
  declare "hd_key" long varchar;
  declare "hd_entry" long varchar;
  declare "pwd" long varchar;
  declare "first" long varchar;
  declare "last" long varchar;
  declare "xpath" long varchar;
  declare "authinfo" long varchar;
  declare "namespace" long varchar;
  declare "mustUnderstand" long varchar;

header_loop:
loop
  set hd_key = next_soap_header( hd_key );
  if hd_key is NULL then
    // no more header entries.
    leave header_loop;
  end if;
  if hd_key = 'Authentication' then
    set hd_entry = soap_header( hd_key );
    set xpath = '/*:' || hd_key || '/*:userName';
    set "namespace" = soap_header( hd_key, 1, '@namespace' );
    set mustUnderstand = soap_header( hd_key, 1, 'mustUnderstand' );
    begin
    // parse for the pieces that you are interested in
      declare crsr cursor for select * from
        openxml( hd_entry, xpath )
          with ( pwd long varchar '@:pwd',
                "first" long varchar '*:first/text()',
                "last" long varchar '*:last/text()' );
      open crsr;
      fetch crsr into pwd, "first", "last";
      close crsr;
    end;
    // build a response header, based on the pieces from the request header
    set authinfo = XMLELEMENT( 'Validation',
      XMLATTRIBUTES(
        "namespace" as xmlns,
        "mustUnderstand" as mustUnderstand ),
      XMLELEMENT( 'first', "first" ),
      XMLELEMENT( 'last', "last" ) );

      call sa_set_soap_header( 'authinfo', authinfo);
    end if;
end loop header_loop;
// code to validate user/session and check item goes here...

```

```

        select count, item, 'available';
    end;

```

The following client-side code illustrates how to create SOAP procedures and functions that send parameters and SOAP headers. Wrapper procedures are used to populate the web service procedure calls and process the responses. The **soapAddItemProc** procedure illustrates the use of a SOAP web service procedure, the **soapAddItemFunction** illustrates the use of a SOAP web service function, and the **httpAddItemFunction** function illustrates how a SOAP payload may be passed to an HTTP web service procedure.

```

/* -- SOAP client procedure --
   uses substitution parameters to send SOAP headers
   a single inout parameter is used to receive SOAP headers
*/
create or replace procedure soapAddItemProc("amount" int, item long varchar,
inout inoutheader long varchar, in inheader long varchar )
    url 'http://localhost/store'
    set 'SOAP( OP=addItems )'
    type 'SOAP:DOC'
    soapheader '!inoutheader!inheader';

/* Wrapper that calls soapAddItemProc
   demonstrates:
       how to send and receive soap headers
       how to send parameters
*/
create or replace procedure addItemProcWrapper( amount int, item long
varchar, "first" long varchar, "last" long varchar )
    begin
        declare io_header long varchar;    // inout (write/read) soap header
        declare resxml long varchar;
        declare soap_header_sent long varchar;
        declare i_header long varchar;    // in (write) only soap header
        declare err int;
        declare crsr cursor for call soapAddItemProc( amount, item,
io_header, i_header );

        set io_header = XMLELEMENT( 'Authentication',
            XMLATTRIBUTES('CustomerOrderURN' as xmlns),
            XMLELEMENT('userName', XMLATTRIBUTES(
                'none' as pwd,
                '1' as mustUnderstand ),
                XMLELEMENT( 'first', "first" ),
                XMLELEMENT( 'last', "last" ) ) );
        set i_header = '<Session xmlns="SomeSession">123456789</Session>';
        set soap_header_sent = io_header || i_header;
        open crsr;
        fetch crsr into resxml, err;
        close crsr;

        select resxml, err, soap_header_sent, io_header as
soap_header_received;
    end;
/* example call to addItemProcWrapper */
call addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );

/* -- SOAP client function --
   uses substitution parameters to send SOAP headers
   Entire SOAP response envelope is returned. SOAP headers can be parsed
   using openxml

```

```

*/
create or replace function soapAddItemFunction("amount" int, item long
varchar, in inheader1 long varchar, in inheader2 long varchar )
returns XML
    url 'http://localhost/store'
    set 'SOAP( OP=addItems )'
    type 'SOAP:DOC'
    soapheader '!inheader1!inheader2';

/* Wrapper that calls soapAddItemFunction
demonstrates the use of SOAP function:
    how to send and receive soap headers
    how to send parameters and process response
*/
create or replace procedure addItemFunctionWrapper( amount int, item long
varchar, "first" long varchar, "last" long varchar )
begin
    declare i_header1 long varchar;
    declare i_header2 long varchar;
    declare res long varchar;
    declare ns long varchar;
    declare xpath long varchar;
    declare header_entry long varchar;
    declare localname long varchar;
    declare namespaceuri long varchar;
    declare r_quantity int;
    declare r_item long varchar;
    declare r_status long varchar;

    set i_header1 = XMLELEMENT( 'Authentication',
        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
        XMLELEMENT('userName', XMLATTRIBUTES(
            'none' as pwd,
            '1' as mustUnderstand ),
            XMLELEMENT( 'first', "first" ),
            XMLELEMENT( 'last', "last" ) ) );
    set i_header2 = '<Session xmlns="SessionURN">123456789</Session>';

    set res = soapAddItemFunction( amount, item, i_header1, i_header2 );

    set ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" '
    | ' xmlns:mp="urn:iAnywhere-com:sa-xpath-metaprop" '
    | ' xmlns:customer="CustomerOrderURN" '
    | ' xmlns:session="SessionURN" '
    | ' xmlns:tns="http://localhost"></ns>';

    // Process headers...
    set xpath = '//SOAP-ENV:Header/*';
    begin
        declare crsr cursor for select * from
            openxml( res, xpath, 1, ns )
                with ( "header_entry" long varchar '@mp:xmltext',
                    "localname" long varchar '@mp:localname',
                    "namespaceuri" long varchar '@mp:namespaceuri' );
        open crsr;
        fetch crsr into "header_entry", "localname", "namespaceuri";
        close crsr;
    end;

    // Process body...
    set xpath = '//tns:row';
    begin

```

```

        declare csrsl cursor for select * from
            openxml( res, xpath, 1, ns )
                with ( "r_quantity" int 'tns:quantity/text()',
                    "r_item" long varchar 'tns:item/text()',
                    "r_status" long varchar 'tns:status/text()' );
        open csrsl;
        fetch csrsl into "r_quantity", "r_item", "r_status";
        close csrsl;
    end;

    select r_item, r_quantity, r_status, header_entry, localname,
    namespaceuri;
    end;

/* example call to addItemFunctionWrapper */
call addItemFunctionWrapper( 5, 'shirt', 'John', 'Smith' );

/*
Demonstrate how a HTTP:POST can be used as a transport for a SOAP payload
Rather than creating a webservice client SOAP procedure, this approach
creates a webservice HTTP procedure that transports a SOAP payload
*/
create or replace function httpAddItemFunction("soapPayload" xml )
returns XML
    url 'http://localhost/store'
    type 'HTTP:POST:text/xml'
    header 'SOAPAction: "http://localhost/addItems"';

/* Wrapper that calls soapAddItemFunction
demonstrates the use of SOAP function:
    how to send and receive soap headers
    how to send parameters and process response
*/

create or replace procedure addItemHttpWrapper( amount int, item long
varchar )
result(response xml)
begin
    declare payload xml;
    declare response xml;

    set payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:m="http://localhost">
    <SOAP-ENV:Body>
        <m:addItems>
            <m:amount>' || amount || '</m:amount>
            <m:item>' || item || '</m:item>
        </m:addItems>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';
    set response = httpAddItemFunction( payload );
    /* process response as demonstrated in addItemFunctionWrapper */
    select response;
end
go

/* example call to addItemHttpWrapper */
call addItemHttpWrapper( 5, 'shirt' );

```

**See also**

- [“SOAP request header management” on page 777](#)
- [“Accessing client-supplied SOAP request headers” on page 749](#)

---



---

# Three-tier computing and distributed transactions

You can use SQL Anywhere as a database server or **resource manager**, participating in distributed transactions coordinated by a transaction server.

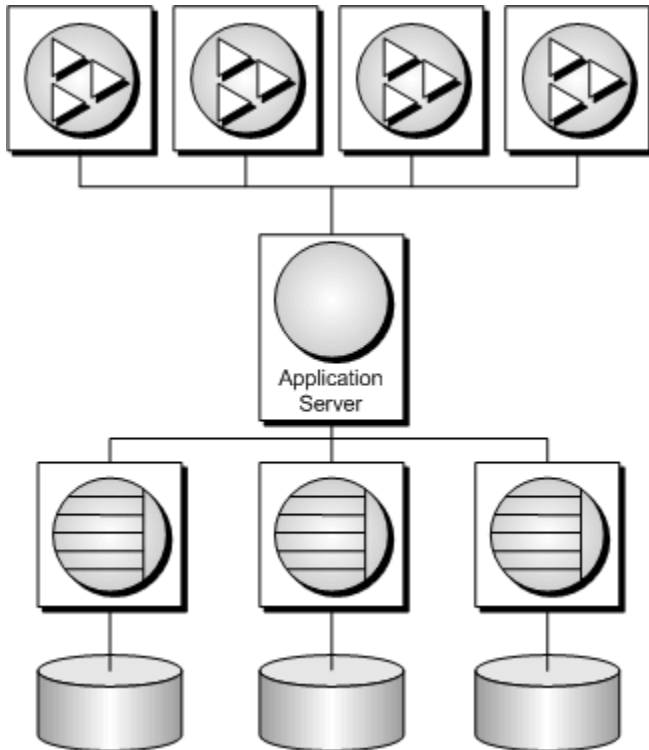
A three-tier environment, where an application server sits between client applications and a set of resource managers, is a common distributed-transaction environment. Sybase EAServer and some other application servers are also transaction servers.

Sybase EAServer and Microsoft Transaction Server both use the Microsoft Distributed Transaction Coordinator (DTC) to coordinate transactions. SQL Anywhere provides support for distributed transactions controlled by the DTC service, so you can use SQL Anywhere with either of these application servers, or any other product based on the DTC model.

When integrating SQL Anywhere into a three-tier environment, most of the work needs to be done from the application server. This section provides an introduction to the concepts and architecture of three-tier computing, and an overview of relevant SQL Anywhere features. It does not describe how to configure your application server to work with SQL Anywhere. For more information, see your application server documentation.

## Three-tier computing architecture

In three-tier computing, application logic is held in an application server, such as Sybase EAServer, which sits between the resource manager and the client applications. In many situations, a single application server may access multiple resource managers. In the Internet case, client applications are browser-based, and the application server is generally a web server extension.



Sybase EAServer stores application logic in the form of components, and makes these components available to client applications. The components may be PowerBuilder components, JavaBeans, or COM components.

For more information, see your Sybase EAServer documentation.

## Distributed transactions in three-tier computing

When client applications or application servers work with a single transaction processing database, such as SQL Anywhere, there is no need for transaction logic outside the database itself, but when working with multiple resource managers, transaction control must span the resources involved in the transaction. Application servers provide transaction logic to their client applications—guaranteeing that sets of operations are executed atomically.

Many transaction servers, including Sybase EAServer, use the Microsoft Distributed Transaction Coordinator (DTC) to provide transaction services to their client applications. DTC uses **OLE transactions**, which in turn use the **two-phase commit** protocol to coordinate transactions involving multiple resource managers. You must have DTC installed to use the features described in this section.

### SQL Anywhere in distributed transactions

SQL Anywhere can take part in transactions coordinated by DTC, which means that you can use SQL Anywhere databases in distributed transactions using a transaction server such as Sybase EAServer or

Microsoft Transaction Server. You can also use DTC directly in your applications to coordinate transactions across multiple resource managers.

## The vocabulary of distributed transactions

This section assumes some familiarity with distributed transactions. For information, see your transaction server documentation. This section describes some commonly used terms.

- **Resource managers** are those services that manage the data involved in the transaction.

The SQL Anywhere database server can act as a resource manager in a distributed transaction when accessed through ADO.NET, OLE DB, or ODBC. The SQL Anywhere .NET Data Provider, OLE DB provider, and ODBC driver act as resource manager proxies on the client computer. The SQL Anywhere .NET Data Provider supports distributed transactions using DbProviderFactory and TransactionScope.

- Instead of communicating directly with the resource manager, application components can communicate with **resource dispensers**, which in turn manage connections or pools of connections to the resource managers.

SQL Anywhere supports two resource dispensers: the ODBC driver manager and OLE DB.

- When a transactional component requests a database connection (using a resource manager), the application server **enlists** each database connection that takes part in the transaction. DTC and the resource dispenser perform the enlistment process.

### Two-phase commit

Distributed transactions are managed using two-phase commit. When the work of the transaction is complete, the transaction manager (DTC) asks all the resource managers enlisted in the transaction whether they are ready to commit the transaction. This phase is called **preparing** to commit.

If all the resource managers respond that they are prepared to commit, DTC sends a commit request to each resource manager, and responds to its client that the transaction is completed. If one or more resource manager does not respond, or responds that it cannot commit the transaction, all the work of the transaction is rolled back across all resource managers.

## How application servers use DTC

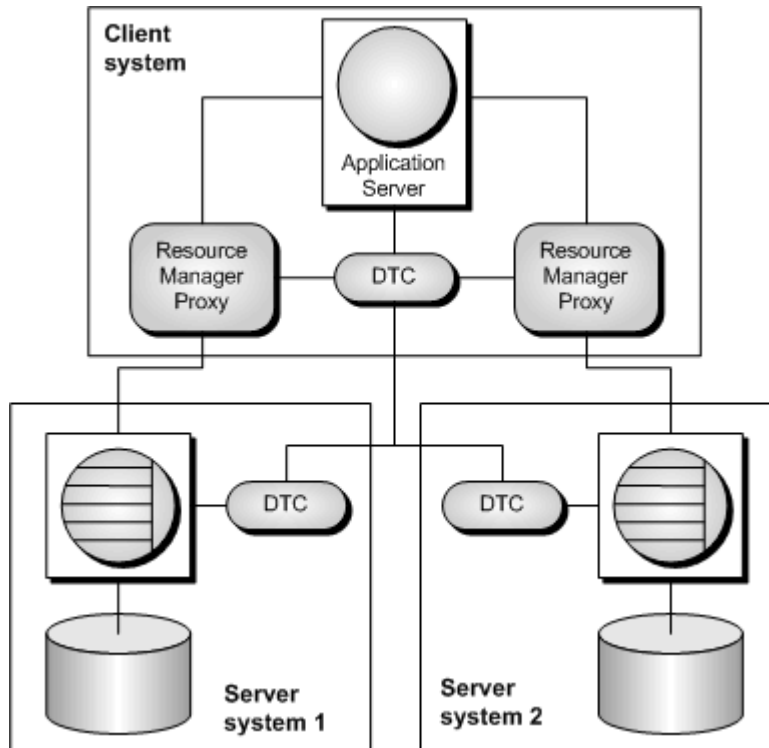
Sybase EAServer and Microsoft Transaction Server are both component servers. The application logic is held in the form of components, and made available to client applications.

Each component has a transaction attribute that indicates how the component participates in transactions. When building the component, you must program the work of the transaction into the component—the resource manager connections, the operations on the data for which each resource manager is responsible. However, you do not need to add transaction management logic to the component. Once the transaction

attribute is set, to indicate that the component needs transaction management, EAServer uses DTC to enlist the transaction and manage the two-phase commit process.

## Distributed transaction architecture

The following diagram illustrates the architecture of distributed transactions. In this case, the resource manager proxy is either ADO.NET, OLE DB, or ODBC.



In this case, a single resource dispenser is used. The application server asks DTC to prepare a transaction. DTC and the resource dispenser enlist each connection in the transaction. Each resource manager must be in contact with both the DTC and the database, so the work can be performed and the DTC can be notified of its transaction status when required.

A Distributed Transaction Coordinator (DTC) service must be running on each computer to operate distributed transactions. You can start or stop DTC from the Microsoft Windows **Services** window; the DTC service task is named **MSDTC**.

For more information, see your DTC or EAServer documentation.

## Distributed transactions

While SQL Anywhere is enlisted in a distributed transaction, it hands transaction control over to the transaction server, and SQL Anywhere ensures that it does not perform any implicit transaction management. The following conditions are imposed automatically by SQL Anywhere when it participates in distributed transactions:

- Autocommit is automatically turned off, if it is in use.
- Data definition statements (which commit as a side effect) are disallowed during distributed transactions.
- An explicit COMMIT or ROLLBACK issued by the application directly to SQL Anywhere, instead of through the transaction coordinator, generates an error. The transaction is not aborted, however.
- A connection can participate in only a single distributed transaction at a time.
- There must be no uncommitted operations at the time the connection is enlisted in a distributed transaction.

## DTC isolation levels

DTC has a set of isolation levels, which the application server specifies. These isolation levels map to SQL Anywhere isolation levels as follows:

DTC isolation level	SQL Anywhere isolation level
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLEREAD	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

## Recovery from distributed transactions

If the database server faults while uncommitted operations are pending, it must either rollback or commit those operations on startup to preserve the atomic nature of the transaction.

If uncommitted operations from a distributed transaction are found during recovery, the database server attempts to connect to DTC and requests that it be re-enlisted in the pending or in-doubt transactions. Once the re-enlistment is complete, DTC instructs the database server to roll back or commit the outstanding operations.

If the reenlistment process fails, SQL Anywhere has no way of knowing whether the in-doubt operations should be committed or rolled back, and recovery fails. If you want the database in such a state to recover, regardless of the uncertain state of the data, you can force recovery using the following database server options:

- **-tmf** If DTC cannot be located, the outstanding operations are rolled back and recovery continues.
- **-tmt** If re-enlistment is not achieved before the specified time, the outstanding operations are rolled back and recovery continues.

### See also

- “-tmf dbeng12/dbsrv12 server option” [[SQL Anywhere Server - Database Administration](#)]
- “-tmt dbeng12/dbsrv12 server option” [[SQL Anywhere Server - Database Administration](#)]

---

# Database tools interface (DBTools)

SQL Anywhere includes Sybase Central and a set of utilities for managing databases. These database management utilities perform tasks such as backing up databases, creating databases, translating transaction logs to SQL, and so on.

## Supported platforms

All the database management utilities use a shared library called the **database tools library**. It is supplied for Windows operating systems and for Linux, Unix, and Mac OS X. For Windows, the name of this library is *dbtool12.dll*. For Linux and Unix, the name of this library is *libdbtool12\_r.so*. For Mac OS X, the name of this library is *libdbtool12\_r.dylib*.

You can develop your own database management utilities or incorporate database management features into your applications by calling the database tools library. This section describes the interface to the database tools library. This section assumes you are familiar with how to call library routines from the development environment you are using.

The database tools library has functions, or entry points, for each of the database management utilities. In addition, functions must be called before use of other database tools functions and when you have finished using other database tools functions.

## Windows Mobile

The *dbtool12.dll* library is supplied for Windows Mobile, but includes only entry points for DBToolsInit, DBToolsFini, DBRemoteSQL, and DBSynchronizeLog. Other entry points are not provided for Windows Mobile.

## The *dbtools.h* header file

The *dbtools* header file lists the entry points to the DBTools library and also the structures used to pass information to and from the library. The *dbtools.h* file is installed into the *SDK\Include* subdirectory under your SQL Anywhere installation directory. You should consult the *dbtools.h* file for the latest information about the entry points and structure members.

The *dbtools.h* header file includes other files such as:

- ***sqlca.h*** This is included for resolution of various macros, not for the SQLCA itself.
- ***dllapi.h*** Defines preprocessor macros for operating-system dependent and language-dependent macros.
- ***dbtlvers.h*** Defines the `DB_TOOLS_VERSION_NUMBER` preprocessor macro and other version specific macros.

## The *sqldef.h* header file

The *sqldef.h* header file includes error return values.

### The *dbrmt.h* header file

The *dbrmt.h* header file included with SQL Anywhere describes the DBRemoteSQL entry point in the DBTools library and also the structure used to pass information to and from the DBRemoteSQL entry point. The *dbrmt.h* file is installed into the *SDK\Include* subdirectory under your SQL Anywhere installation directory. You should consult the *dbrmt.h* file for the latest information about the DBRemoteSQL entry point and structure members.

## DBTools import libraries

To use the DBTools functions, you must link your application against a DBTools **import library** that contains the required function definitions.

For Unix systems, no import library is required. Link directly against *libdbtool12.so* (non-threaded) or *libdbtool12\_r.so* (threaded).

### Import libraries

Import libraries for the DBTools interface are provided with SQL Anywhere for Windows and Windows Mobile. For Windows, they can be found in the *SDK\Lib\x86* and *SDK\Lib\x64* subdirectories under your SQL Anywhere installation directory. For Windows Mobile, the import library can be found in the *SDK\Lib\CE\Arm.50* subdirectory under your SQL Anywhere installation directory. The provided DBTools import libraries are as follows:

Compiler	Library
Microsoft Windows	<i>dbtlstm.lib</i>
Microsoft Windows Mobile	<i>dbtool12.lib</i>

## Initializing and finalizing the DBTools library

Before using any other DBTools functions, you must call `DBToolsInit`. When you are finished using the DBTools library, you must call `DBToolsFini`.

The primary purpose of the `DBToolsInit` and `DBToolsFini` functions is to allow the DBTools library to load and unload the SQL Anywhere message library. The message library contains localized versions of all error messages and prompts that DBTools uses internally. If `DBToolsFini` is not called, the reference count of the messages library is not decremented and it will not be unloaded, so be careful to ensure there is a matched pair of `DBToolsInit`/`DBToolsFini` calls.

The following code fragment illustrates how to initialize and finalize DBTools:

```
// Declarations
a_dbtools_info info;
short          ret;
```



```

//Initialize the a_dbtools_info structure
memset( &info, 0, sizeof( a_dbtools_info ) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize the DBTools library
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
    // library initialization failed
    ...
}
// call some DBTools routines ...
...
// finalize the DBTools library
DBToolsFini( &info );

```

## Calling the DBTools functions

All the tools are run by first filling out a structure, and then calling a function (or **entry point**) in the DBTools library. Each entry point takes a pointer to a single structure as argument.

The following example shows how to use the DBBackup function on a Windows operating system.

```

// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "c:\\\\backup";
backup_info.connectparms = "UID=DBA;PWD=sql;DBF=demo.db";

backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );

```

### See also

- [“DBTools structures” on page 852](#)

## Callback functions

Several elements in DBTools structures are of type MSG\_CALLBACK. These are pointers to callback functions.

### Uses of callback functions

Callback functions allow DBTools functions to return control of operation to the user's calling application. The DBTools library uses callback functions to handle messages sent to the user by the DBTools functions for four purposes:

- **Confirmation** Called when an action needs to be confirmed by the user. For example, if the backup directory does not exist, the tools library asks if it needs to be created.
- **Error message** Called to handle a message when an error occurs, such as when an operation is out of disk space.
- **Information message** Called for the tools to display some message to the user (such as the name of the current table being unloaded).
- **Status information** Called for the tools to display the status of an operation (such as the percentage done when unloading a table).

### Assigning a callback function to a structure

You can directly assign a callback routine to the structure. The following statement is an example using a backup structure:

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG\_CALLBACK is defined in the *dllapi.h* header file supplied with SQL Anywhere. Tools routines can call back to the calling application with messages that should appear in the appropriate user interface, whether that be a windowing environment, standard output on a character-based system, or other user interface.

### Confirmation callback function example

The following example confirmation routine asks the user to answer YES or NO to a prompt and returns the user's selection:

```
extern short _callback ConfirmRtn(
    char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
            "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
    return( ret == IDYES );
}
```

### Error callback function example

The following is an example of an error message handling routine, which displays the error message in a window.

```
extern short _callback ErrorRtn(
    char * errorstr )
{
    if( errorstr != NULL ) {
        MessageBox( HwndParent, errorstr, "Backup Error", MB_ICONSTOP|
            MB_OK );
    }
    return( 0 );
}
```

### Message callback function example

A common implementation of a message callback function outputs the message to the screen:

```
extern short _callback MessageRtn(
    char * messagestr )
{
    if( messagestr != NULL ) {
        OutputMessageToWindow( messagestr );
    }
    return( 0 );
}
```

### Status callback function example

A status callback routine is called when a tool needs to display the status of an operation (like the percentage done unloading a table). A common implementation would just output the message to the screen:

```
extern short _callback StatusRtn(
    char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    }
    return( 0 );
}
```

## Version numbers and compatibility

Each structure has a member that indicates the version number. You should set the version field to the version number of the DBTools library that your application was developed against before calling any DBTools function. The current version of the DBTools library is defined when you include the *dbtools.h* header file.

The following example assigns the current version to an instance of the *a\_backup\_db* structure:

```
backup_info.version = DB_TOOLS_VERSION_NUMBER;
```

The version number allows your application to continue working with newer versions of the DBTools library. The DBTools functions use the version number supplied by your application to allow the application to work, even if new members have been added to the DBTools structure.

When any of the DBTools structures are updated, or when a newer version of the software is released, the version number is augmented. If you use *DB\_TOOLS\_VERSION\_NUMBER* and you rebuild your application with a new version of the DBTools header file, then you must deploy a new version of the DBTools library.

## Bit fields

Many of the DBTools structures use bit fields to hold Boolean information in a compact manner. For example, the backup structure includes the following bit fields:

```
a_bit_field    backup_database : 1;
a_bit_field    backup_logfile  : 1;
a_bit_field    no_confirm     : 1;
a_bit_field    quiet          : 1;
```

```
a_bit_field    rename_log    : 1;
a_bit_field    truncate_log  : 1;
a_bit_field    rename_local_log: 1;
a_bit_field    server_backup  : 1;
```

Each bit field is one bit long, indicated by the 1 to the right of the colon in the structure declaration. The specific data type used depends on the value assigned to `a_bit_field`, which is set at the top of `dbtools.h`, and is operating system-dependent.

You assign a value of 0 or 1 to a bit field to pass Boolean information in the structure.

## A DBTools example

You can find this sample and instructions for compiling it in the `%SQLANY%SAMP12%\SQLAnywhere\DBTools` directory. The sample program itself is in `main.cpp`. The sample illustrates how to use the DBTools library to perform a backup of a database.

```
#define WIN32

#include <stdio.h>
#include <string.h>
#include "windows.h"
#include "sqldef.h"
#include "dbtools.h"
extern short _callback ConfirmCallback( char * str )
{
    if( MessageBox( NULL, str, "Backup",
        MB_YESNO|MB_ICONQUESTION ) == IDYES )
    {
        return 1;
    }
    return 0;
}
extern short _callback MessageCallback( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback StatusCallback( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback ErrorCallback( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
typedef void (CALLBACK *DBTOOLSPROC)( void * );
```

```

typedef short (CALLBACK *DBTOOLSFUNC)( void * );

// Main entry point into the program.
int main( int argc, char * argv[] )
{
    a_dbtools_info  dbt_info;
    a_backup_db     backup_info;
    char            dir_name[ _MAX_PATH + 1];
    char            connect[ 256 ];
    HINSTANCE       hinst;
    DBTOOLSFUNC     dbbackup;
    DBTOOLSFUNC     dbtoolsinit;
    DBTOOLSPROC     dbtoolsfini;
    short           ret_code;

    // Always initialize to 0 so new versions
    // of the structure will be compatible.
    memset( &dbt_info, 0, sizeof( a_dbtools_info ) );
    dbt_info.errorrtn = (MSG_CALLBACK)MessageCallBack;

    memset( &backup_info, 0, sizeof( a_backup_db ) );
    backup_info.version = DB_TOOLS_VERSION_NUMBER;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.confirmrtn = (MSG_CALLBACK)ConfirmCallBack;
    backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
    backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
    backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;
    if( argc > 1 )
    {
        strncpy( dir_name, argv[1], _MAX_PATH );
    }
    else
    {
        // DBTools does not expect (or like) a trailing slash
        strcpy( dir_name, "c:\\temp" );
    }
    backup_info.output_dir = dir_name;
    if( argc > 2 )
    {
        strncpy( connect, argv[2], 255 );
    }
    else
    {
        strcpy( connect, "DSN=SQL Anywhere 12 Demo" );
    }
    backup_info.connectparms = connect;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.backup_database = 1;
    backup_info.backup_logfile = 1;
    backup_info.rename_log = 0;
    backup_info.truncate_log = 0;
    hinst = LoadLibrary( "dbtool12.dll" );
    if( hinst == NULL )
    {
        // Failed
        return EXIT_FAIL;
    }
    dbbackup = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
        "_DBBackup@4" );
    dbtoolsinit = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
        "_DBToolsInit@4" );
    dbtoolsfini = (DBTOOLSPROC) GetProcAddress( (HMODULE)hinst,

```

```
    "_DBToolsFini@4" );
ret_code = (*dbtoolsinit)( &dbt_info );
if( ret_code != EXIT_OKAY ) {
    return ret_code;
}
ret_code = (*dbbackup)( &backup_info );
(*dbtoolsfini)( &dbt_info );
FreeLibrary( hinst );
return ret_code;
}
```

## SQL Anywhere Database Tools C API reference

### Header files

- `dbtools.h`
- `dbrmt.h`

## DBTools structures

This section lists the structures that are used to exchange information with the DBTools library. The structures are listed alphabetically. With the exception of the `a_remote_sql` structure, all of these structures are defined in `dbtools.h`. The `a_remote_sql` structure is defined in `dbrmt.h`.

Many of the structure elements correspond to command line options on the corresponding utility. For example, several structures have a member named `quiet`, which can take on values of 0 or 1. This member corresponds to the quiet operation (`-q`) option used by many of the utilities.

The data structures defined in this file can be used with the DBTools API for the major version of SQL Anywhere to which the file applies. Applications built using, for example, the version 10.0.0 `dbtools.h` file will NOT be able to access `dbtool9.dll` or `dbtool11.dll`.

Within a major version, any changes to the structures will be made in such a way that applications built with earlier or later versions of `dbtools.h` for the same major version will work. Applications built with earlier versions will not have access to new fields, so DBTools will provide default values which give the same behavior as in earlier versions. Generally, this means new fields added in point releases appear at the end of a structure.

The behavior for applications built with later versions within the same major release depends on the value provided in the "version" field of the structure. If the version number provided corresponds to an earlier version, the application can call an earlier version of the DBTools DLL just as if the application was built using that version of `dbtools.h`. If the version number provided is the current version, attempting to use an earlier version of the DBTools DLL will result in an error.

See `dbtlvers.h` for version number defines.

## DBBackup method

Backs up a database.

### Syntax

```
_crtn short _entry DBBackup(const a_backup_db * pdb)
```

### Parameters

- **pdb** Pointer to a properly initialized a\_backup\_db structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the dbbackup utility.

The DBBackup function manages all client-side database backup tasks. For a description of these tasks, see [“Backup utility \(dbbackup\)” \[SQL Anywhere Server - Database Administration\]](#).

To perform a server-side backup, use the BACKUP DATABASE statement.

### See also

- [“a\\_backup\\_db structure \[database tools\]” on page 867](#)
- [“BACKUP statement” \[SQL Anywhere Server - SQL Reference\]](#)

## DBChangeLogName method

Changes the name of the transaction log file.

### Syntax

```
_crtn short _entry DBChangeLogName(const a_change_log * pcl)
```

### Parameters

- **pcl** Pointer to a properly initialized a\_change\_log structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the dblog utility.

The -t option of the Transaction Log utility (dblog) changes the name of the transaction log. DBChangeLogName provides a programmatic interface to this function.

### See also

- [“a\\_change\\_log structure \[database tools\]” on page 869](#)
- [“Transaction Log utility \(dblog\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“ALTER DATABASE statement” \[SQL Anywhere Server - SQL Reference\]](#)

## DBCcreate method

Creates a database.

### Syntax

```
_crtn short _entry DBCcreate(a_create_db * pcdb)
```

### Parameters

- **pcdb** Pointer to a properly initialized a\_create\_db structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the dbinit utility.

### See also

- [“a\\_create\\_db structure \[database tools\]” on page 871](#)
- [“Initialization utility \(dbinit\)” \[SQL Anywhere Server - Database Administration\]](#)

## DBCcreatedVersion method

Determines the version of SQL Anywhere that was used to create a database file, without attempting to start the database.

### Syntax

```
_crtn short _entry DBCcreatedVersion(a_db_version_info * pdvi)
```

### Parameters

- **pdvi** Pointer to a properly initialized a\_db\_version\_info structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

Currently, this function only differentiates between databases built with version 9 or earlier and those built with version 10 or later.



Version information is not set if a failing code is returned.

**See also**

- [“a\\_db\\_version\\_info structure \[database tools\]” on page 877](#)
- [“Version enumeration \[database tools\]” on page 867](#)
- [“CREATE DATABASE statement” \[SQL Anywhere Server - SQL Reference\]](#)

## DBErase method

Erases a database file and/or transaction log file.

**Syntax**

```
_crtn short _entry DBErase(const an_erase_db * pedb)
```

**Parameters**

- **pedb** Pointer to a properly initialized an\_erase\_db structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

This function is used by the dberase utility.

**See also**

- [“an\\_erase\\_db structure \[database tools\]” on page 905](#)
- [“Erase utility \(dberase\)” \[SQL Anywhere Server - Database Administration\]](#)

## DBInfo method

Returns information about a database file.

**Syntax**

```
_crtn short _entry DBInfo(a_db_info * pdbi)
```

**Parameters**

- **pdbi** Pointer to a properly initialized a\_db\_info structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

This function is used by the dbinfo utility.

### See also

- [“a\\_db\\_info structure \[database tools\]” on page 874](#)
- [“DBInfoDump method \[database tools\]” on page 856](#)
- [“DBInfoFree method \[database tools\]” on page 856](#)
- [“Information utility \(dbinfo\)” \[SQL Anywhere Server - Database Administration\]](#)

## DBInfoDump method

Returns information about a database file.

### Syntax

```
_crtn short _entry DBInfoDump(a_db_info * pdbi)
```

### Parameters

- **pdbi** Pointer to a properly initialized a\_db\_info structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the dbinfo utility when the -u option is used.

### See also

- [“a\\_db\\_info structure \[database tools\]” on page 874](#)
- [“DBInfo method \[database tools\]” on page 855](#)
- [“DBInfoFree method \[database tools\]” on page 856](#)

## DBInfoFree method

Frees resources after the DBInfoDump function is called.

### Syntax

```
_crtn short _entry DBInfoFree(a_db_info * pdbi)
```

### Parameters

- **pdbi** Pointer to a properly initialized a\_db\_info structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

**See also**

- [“a\\_db\\_info structure \[database tools\]” on page 874](#)
- [“DBInfo method \[database tools\]” on page 855](#)
- [“DBInfoDump method \[database tools\]” on page 856](#)

## DBLicense method

Modifies or reports the licensing information of the database server.

**Syntax**

```
_crtn short _entry DBLicense(const a_dblic_info * pdi)
```

**Parameters**

- **pdi** Pointer to a properly initialized a\_dblic\_info structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**See also**

- [“a\\_dblic\\_info structure \[database tools\]” on page 877](#)
- [“Server Licensing utility \(dblic\)” \[SQL Anywhere Server - Database Administration\]](#)

## DBLogFileInfo method

Returns the log file and mirror log file paths of a non-running database file.

**Syntax**

```
_crtn short _entry DBLogFileInfo(const a_log_file_info * plfi)
```

**Parameters**

- **plfi** Pointer to a properly initialized a\_log\_file\_info structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

Note that this function will only work for databases that have been created with SQL Anywhere 10.0.0 and up.

**See also**

- [“a\\_log\\_file\\_info structure \[database tools\]” on page 879](#)

## DBRemoteSQL method

Accesses the SQL Remote Message Agent.

### Syntax

```
_crtn short _entry DBRemoteSQL(a_remote_sql * prs)
```

### Parameters

- **prs** Pointer to a properly initialized a\_remote\_sql structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

For information about the features you can access, see [“SQL Remote Message Agent utility \(dbremote\)” \[SQL Remote\]](#).

### See also

- [“a\\_remote\\_sql structure \[database tools\]” on page 880](#)
- [“SQL Remote systems” \[SQL Remote\]](#)

## DBSynchronizeLog method

Synchronize a database with a MobiLink server.

### Syntax

```
_crtn short _entry DBSynchronizeLog(const a_sync_db * psdb)
```

### Parameters

- **psdb** Pointer to a properly initialized a\_sync\_db structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### See also

- [“a\\_sync\\_db structure \[database tools\]” on page 887](#)
- [“DBTools interface for dbmlsync” \[MobiLink - Client Administration\]](#)

## DBToolsFini method

Decrements a reference counter and frees resources when an application is finished with the DBTools library.

**Syntax**

```
_crtn short _entry DBToolsFini(const a_dbtools_info * pdi)
```

**Parameters**

- **pdi** Pointer to a properly initialized a\_dbtools\_info structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

The DBToolsFini function must be called at the end of any application that uses the DBTools interface. Failure to do so can lead to lost memory resources.

**See also**

- [“a\\_dbtools\\_info structure \[database tools\]” on page 878](#)
- [“DBToolsInit method \[database tools\]” on page 859](#)

## DBToolsInit method

Prepares the DBTools library for use.

**Syntax**

```
_crtn short _entry DBToolsInit(const a_dbtools_info * pdi)
```

**Parameters**

- **pdi** Pointer to a properly initialized a\_dbtools\_info structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

The primary purpose of the DBToolsInit function is to load the SQL Anywhere messages library. The messages library contains localized versions of error messages and prompts used by the functions in the DBTools library.

The DBToolsInit function must be called at the start of any application that uses the DBTools interface, before any other DBTools functions.

**See also**

- [“a\\_dbtools\\_info structure \[database tools\]” on page 878](#)
- [“DBToolsFini method \[database tools\]” on page 858](#)
- [“A DBTools example” on page 850](#)

## DBToolsVersion method

Returns the version number of the DBTools library.

### Syntax

```
_crtn short _entry DBToolsVersion(void)
```

### Remarks

Use the DBToolsVersion function to check that the DBTools library is not older than one against which your application is developed. While applications can run against newer versions of DBTools, they cannot run against older versions.

### See also

- [“Version numbers and compatibility” on page 849](#)

## DBTranslateLog method

Translates a transaction log file to SQL.

### Syntax

```
_crtn short _entry DBTranslateLog(const a_translate_log * ptl)
```

### Parameters

- **ptl** Pointer to a properly initialized a\_translate\_log structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the dbtran utility.

### See also

- [“a\\_translate\\_log structure \[database tools\]” on page 898](#)
- [“Log Translation utility \(dbtran\)” \[SQL Anywhere Server - Database Administration\]](#)

## DBTruncateLog method

Truncates a transaction log file.

### Syntax

```
_crtn short _entry DBTruncateLog(const a_truncate_log * ptl)
```

**Parameters**

- **ptl** Pointer to a properly initialized `a_truncate_log` structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

This function is used by the `dbbackup` utility.

**See also**

- [“a\\_truncate\\_log structure \[database tools\]” on page 902](#)
- [“Backup utility \(dbbackup\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“BACKUP statement” \[SQL Anywhere Server - SQL Reference\]](#)

## DBUnload method

Unloads a database.

**Syntax**

```
_crtn short _entry DBUnload(an_unload_db * pldb)
```

**Parameters**

- **pldb** Pointer to a properly initialized `an_unload_db` structure.

**Returns**

Return code, as listed in [“Software component exit codes” on page 912](#).

**Remarks**

This function is used by the `dbunload` and `dbxtract` utilities.

**See also**

- [“an\\_unload\\_db structure \[database tools\]” on page 906](#)
- [“Unload utility \(dbunload\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“Extraction utility \(dbxtract\)” \[SQL Remote\]](#)

## DBUpgrade method

Upgrades a database file.

**Syntax**

```
_crtn short _entry DBUpgrade(const an_upgrade_db * pldb)
```

### Parameters

- **puadb** Pointer to a properly initialized `an_upgrade_db` structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the `dbupgrad` utility.

### See also

- [“an\\_upgrade\\_db structure \[database tools\]” on page 911](#)
- [“Upgrade utility \(dbupgrad\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“ALTER DATABASE statement” \[SQL Anywhere Server - SQL Reference\]](#)

## DBValidate method

Validates all or part of a database.

### Syntax

```
_crtn short _entry DBValidate(const a_validate_db * pvdb)
```

### Parameters

- **pvdb** Pointer to a properly initialized `a_validate_db` structure.

### Returns

Return code, as listed in [“Software component exit codes” on page 912](#).

### Remarks

This function is used by the `dbvalid` utility.

Caution: Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, spurious errors may be reported indicating some form of database corruption even though no corruption actually exists.

### See also

- [“a\\_validate\\_db structure \[database tools\]” on page 903](#)
- [“Validation utility \(dbvalid\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“VALIDATE statement” \[SQL Anywhere Server - SQL Reference\]](#)

## Checkpoint enumeration

Used in the `a_backup_db` structure to control copying of the checkpoint log.



**Syntax**enum **Checkpoint****Members**

Member name	Description	Value
BACKUP_CHKPT_LOG_COPY	Use to generate WITH CHECKPOINT LOG COPY clause.	0
BACKUP_CHKPT_LOG_NOCOPY	Use to generate WITH CHECKPOINT LOG NO-COPY clause.	
BACKUP_CHKPT_LOG_RECOVER	Use to generate WITH CHECKPOINT LOG RECOVER clause.	
BACKUP_CHKPT_LOG_AUTO	Use to generate WITH CHECKPOINT LOG AUTO clause.	
BACKUP_CHKPT_LOG_DEFAULT	Use to omit WITH CHECKPOINT clause.	

**See also**

- [“a\\_backup\\_db structure \[database tools\]” on page 867](#)
- [“DBBackup method \[database tools\]” on page 853](#)

## Padding enumeration

Blank padding enumeration specifies the blank\_pad setting in a\_create\_db.

**Syntax**enum **Padding****Members**

Member name	Description
NO_BLANK_PADDING	Does not use blank padding.
BLANK_PADDING	Uses blank padding.

**See also**

- [“a\\_create\\_db structure \[database tools\]” on page 871](#)
- [“DBCreate method \[database tools\]” on page 854](#)

## Unit enumeration

Used in the `a_create_db` structure, to specify the value of `db_size_unit`.

### Syntax

```
enum Unit
```

### Members

Member name	Description
DBSP_UNIT_NONE	Units not specified.
DBSP_UNIT_PAGES	Size is specified in pages.
DBSP_UNIT_BYTES	Size is specified in bytes.
DBSP_UNIT_KILOBYTES	Size is specified in kilobytes.
DBSP_UNIT_MEGABYTES	Size is specified in megabytes.
DBSP_UNIT_GIGABYTES	Size is specified in gigabytes.
DBSP_UNIT_TERABYTES	Size is specified in terabytes.

### See also

- [“a\\_create\\_db structure \[database tools\]” on page 871](#)
- [“DBCreate method \[database tools\]” on page 854](#)

## Unload enumeration

The type of unload being performed, as used by the `an_unload_db` structure.

### Syntax

```
enum Unload
```

### Members

Member name	Description
UNLOAD_ALL	Unload both data and schema.
UNLOAD_DATA_ONLY	Unload data. Do not unload schema. Equivalent to <code>dbunload -d</code> option.
UNLOAD_NO_DATA	No data. Unload schema only. Equivalent to <code>dbunload -n</code> option.

Member name	Description
UNLOAD_NO_DATA_FULL_SCRIPT	No data. Include LOAD/INPUT statements in reload script. Equivalent to dbunload -nl option.
UNLOAD_NO_DATA_NAME_ORDER	No data. Objects will be output ordered by name.

**See also**

- [“an\\_unload\\_db structure \[database tools\]” on page 906](#)
- [“DBUnload method \[database tools\]” on page 861](#)

## UserList enumeration

The type of a user list, as used by an a\_translate\_log structure.

**Syntax**

```
enum UserList
```

**Members**

Member name	Description
DBTRAN_INCLUDE_ALL	Include operations from all users.
DBTRAN_INCLUDE_SOME	Include operations only from the users listed in the supplied user list.
DBTRAN_EXCLUDE_SOME	Exclude operations from the users listed in the supplied user list.

**See also**

- [“a\\_translate\\_log structure \[database tools\]” on page 898](#)
- [“DBTranslateLog method \[database tools\]” on page 860](#)

## Validation enumeration

The type of validation being performed, as used by the a\_validate\_db structure.

**Syntax**

```
enum Validation
```

**Members**

Member name	Description	Value
VALIDATE_NORMAL	Validate with the default check only.	0
VALIDATE_DATA	(obsolete)	
VALIDATE_INDEX	(obsolete)	
VALIDATE_EXPRESS	Validate with express check. Equivalent to dbvalid -fx option.	
VALIDATE_FULL	(obsolete)	
VALIDATE_CHECKSUM	Validate database checksums. Equivalent to dbvalid -s option.	
VALIDATE_DATABASE	Validate database. Equivalent to dbvalid -d option.	
VALIDATE_COMPLETE	Perform all possible validation activities.	

**See also**

- [“a\\_validate\\_db structure \[database tools\]” on page 903](#)
- [“DBValidate method \[database tools\]” on page 862](#)
- [“Corruption detection using checksums” \[SQL Anywhere Server - Database Administration\]](#)

## Verbosity enumeration

Verbosity enumeration specifies the volume of output.

**Syntax**

```
enum verbosity
```

**Members**

Member name	Description
VB_QUIET	No output.
VB_NORMAL	Normal amount of output.
VB_VERBOSE	Verbose output, useful for debugging.

**See also**

- [“a\\_create\\_db structure \[database tools\]” on page 871](#)
- [“DBCreate method \[database tools\]” on page 854](#)

## Version enumeration

Used in the `a_db_version_info` structure, to indicate the version of SQL Anywhere that initially created the database.

### Syntax

```
enum Version
```

### Members

Member name	Description	Value
VERSION_UNKNOWN	Unable to determine the version of SQL Anywhere that created the database.	0
VERSION_PRE_10	Database was created using SQL Anywhere version 9 or earlier.	9
VERSION_10	Database was created using SQL Anywhere version 10.	10
VERSION_11	Database was created using SQL Anywhere version 11.	11
VERSION_12	Database was created using SQL Anywhere version 12.	12

### See also

- [“a\\_db\\_version\\_info structure \[database tools\]” on page 877](#)
- [“DBCreatedVersion method \[database tools\]” on page 854](#)

## a\_backup\_db structure

Holds the information needed to perform backup tasks using the DBTools library.

### Syntax

```
typedef struct a_backup_db
```

### Members

Member name	Type	Description
backup_database	a_bit_field	Back up the database file. Set TRUE by <code>dbbackup -d</code> option.
backup_interrupted	char	Indicates that the operation was interrupted when non-zero.

Member name	Type	Description
backup_logfile	a_bit_field	Back up the transaction log file. Set TRUE by dbbackup -t option.
chkpt_log_type	char	Control copying of checkpoint log. Must be one of BACKUP_CHKPT_LOG_COPY, BACKUP_CHKPT_LOG_NOCOPY, BACKUP_CHKPT_LOG_RECOVER, BACKUP_CHKPT_LOG_AUTO, or BACKUP_CHKPT_LOG_DEFAULT. Set by dbbackup -k option.
confirmrtn	MSG_CALLBACK	Address of a confirmation request callback routine or NULL.
connectparms	const char *	Parameters needed to connect to the database. They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db". The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe". A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
hotlog_filename	const char *	File name for the live backup file. Set by dbbackup -l option.
msgsrtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
no_confirm	a_bit_field	Operate without confirmation. Set TRUE by dbbackup -y option.
output_dir	const char *	Path to the output directory for backups, for example: "c:\back-up".
page_blocksize	a_sql_uint32	Number of pages in data blocks. If set to 0, then the default is 128. Set by dbbackup -b option.

Member name	Type	Description
progress_messages	a_bit_field	Display progress messages. Set TRUE by dbbackup -p option.
quiet	a_bit_field	Operate without printing messages. Set TRUE by dbbackup -q option.
rename_local_log	a_bit_field	Rename the local backup of the transaction log. Set TRUE by dbbackup -n option.
rename_log	a_bit_field	Rename the transaction log. Set TRUE by dbbackup -r option.
server_backup	a_bit_field	Perform backup on server using BACKUP DATABASE. Set TRUE by dbbackup -s option.
statusrtn	MSG_CALLBACK	Address of a status message callback routine or NULL.
truncate_log	a_bit_field	Delete the transaction log. Set TRUE by dbbackup -x option.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“Checkpoint enumeration \[database tools\]” on page 862](#)
- [“DBBackup method \[database tools\]” on page 853](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## a\_change\_log structure

Holds the information needed to perform dblog tasks using the DBTools library.

**Syntax**

```
typedef struct a_change_log
```

**Members**

Member name	Type	Description
change_logname	a_bit_field	Set TRUE to permit changing of the transaction log name.  Set TRUE by dblog -n or -t option.
change_mirrorname	a_bit_field	Set TRUE to permit changing of the mirror log name.  Set TRUE by dblog -m, -n, or -r option.
dbname	const char *	Database file name.
encryption_key	char *	The encryption key for the database file. Equivalent to dblog -ek or -ep option.
errortrn	MSG_CALL- BACK	Address of an error message callback routine or NULL.
generation_number	unsigned short	The new generation number. Reserved, use zero.
ignore_dbsync_trunc	a_bit_field	When using dbmsync, resets the offset kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed.  Set TRUE by dblog -is option.
ignore_ltm_trunc	a_bit_field	Reserved, use FALSE.
ignore_remote_trunc	a_bit_field	For SQL Remote.  Resets the offset kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed. Set TRUE by dblog -ir option.
logname	const char *	Transaction log file name, or NULL if there is no log.
mirrorname	const char *	The new name of the transaction log mirror file. Equivalent to dblog -m option.
msgtrn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
query_only	a_bit_field	If 1, just display the name of the transaction log. If 0, permit changing of the log name.



Member name	Type	Description
quiet	a_bit_field	Operate without printing messages. Set TRUE by dblog -q option.
set_generation_number	a_bit_field	Reserved. Use FALSE.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).
zap_current_offset	char *	Change the current offset to the specified value.  This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmsync settings. Equivalent to dblog -x option.
zap_starting_offset	char *	Change the starting offset to the specified value.  This is for use only in resetting a transaction log after an unload and reload to match dbremote or dbmsync settings. Equivalent to dblog -z option.

**See also**

- [“DBChangeLogName method \[database tools\]” on page 853](#)

**a\_create\_db structure**

Holds the information needed to create a database using the DBTools library.

**Syntax**

```
typedef struct a_create_db
```

**Members**

Member name	Type	Description
accent_sensitivity	char	One of 'y', 'n', or 'f' (yes, no, French).  Generates one of the ACCENT RESPECT, ACCENT IGNORE or ACCENT FRENCH clauses.

Member name	Type	Description
avoid_view_collisions	a_bit_field	Set TRUE to omit the generation of Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES.  Set TRUE by dbinit -k option.
blank_pad	a_bit_field	Must be one of NO_BLANK_PADDING or BLANK_PADDING.  Treat blanks as significant in string comparisons and hold index information to reflect this. See Blank padding enumeration. Equivalent to dbinit -b option.
case_sensitivity_use_default	a_bit_field	Set TRUE to use the default case sensitivity for the locale.  This only affects UCA. If set TRUE then we do not add the CASE RESPECT clause to the CREATE DATABASE statement.
checksum	a_bit_field	Set to TRUE for ON or FALSE for OFF.  Generates one of CHECKSUM ON or CHECKSUM OFF clauses. Set TRUE by dbinit -s option.
data_store_type	const char *	Reserved. Use NULL.
db_size	unsigned int	When not 0, generates the DATABASE SIZE clause. Equivalent to dbinit -dbs option.
db_size_unit	int	Used with db_size, must be one of DBSP_UNIT_NONE, DBSP_UNIT_PAGES, DBSP_UNIT_BYTES, DBSP_UNIT_KILOBYTES, DBSP_UNIT_MEGABYTES, DBSP_UNIT_GIGABYTES, or DBSP_UNIT_TERABYTES.  When not DBSP_UNIT_NONE, it generates the corresponding keyword (for example, DATABASE SIZE 10 MB is generated when db_size is 10 and db_size_unit is DBSP_UNIT_MEGABYTES). See Database size unit enumeration.
dba_pwd	char *	When not NULL, generates the DBA PASSWORD xxx clause. Equivalent to dbinit -dba option.
dba_uid	char *	When not NULL, generates the DBA USER xxx clause. Equivalent to dbinit -dba option.
dbname	const char *	Database file name.

Member name	Type	Description
default_collation	const char *	The collation for the database. Equivalent to dbinit -z option.
encoding	const char *	The character set encoding. Equivalent to dbinit -ze option.
encrypt	a_bit_field	Set TRUE to generate the ENCRYPTED ON clause or, when encrypted_tables is also set, the ENCRYPTED TABLES ON clause.  Set TRUE by dbinit -e? options.
encrypted_tables	a_bit_field	Set TRUE to encrypt tables.  Used with encrypt, it generates the ENCRYPTED TABLE ON clause instead of the ENCRYPTED ON clause. Set TRUE by dbinit -et option.
encryption_algorithm	const char *	The encryption algorithm (AES, AES256, AES_FIPS, or AES256_FIPS).  Used with encrypt and encryption_key, it generates the ALGORITHM clause. Equivalent to dbinit -ea option.
encryption_key	const char *	The encryption key for the database file.  Used with encrypt, it generates the KEY clause. Equivalent to dbinit -ek option.
errortn	MSG_CALLBACK	Address of an error message callback routine or NULL.
iq_params	void *	Reserved. Use NULL.
jconnect	a_bit_field	Set TRUE to include system procedures needed for jConnect.  Set FALSE by dbinit -i option.
logname	const char *	New transaction log name. Equivalent to dbinit -t option.
mirrorname	const char *	Transaction log mirror name. Equivalent to dbinit -m option.
msgtrtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
nchar_collation	const char *	The NCHAR COLLATION for the database when not NULL. Equivalent to dbinit -zn option.

Member name	Type	Description
page_size	unsigned short	The page size of the database. Equivalent to dbinit -p option.
respect_case	a_bit_field	Make string comparisons case sensitive and hold index information to reflect this.  Set TRUE by dbinit -c option.
startline	const char *	The command line used to start the database server.  For example: "c:\SQLAny12\bin32\dbeng12.exe". If NULL, the default START parameter is "dbeng12 -gp <page_size> -c 10M" where page_size is specified below. Note that "-c 10M" is appended if page_size >= 2048.
verbose	char	See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“Padding enumeration \[database tools\]” on page 863](#)
- [“Unit enumeration \[database tools\]” on page 864](#)
- [“Verbosity enumeration \[database tools\]” on page 866](#)
- [“DBCreate method \[database tools\]” on page 854](#)
- [“Corruption detection using checksums” \[SQL Anywhere Server - Database Administration\]](#)

## a\_db\_info structure

Holds the information needed to return DBInfo information using the DBTools library.

**Syntax**

```
typedef struct a_db_info
```

**Members**

Member name	Type	Description
bit_map_pages	a_sql_uint32	Number of bitmap pages in the database.
charcollations-pecbuffer	char *	Pointer to the char collation string buffer.

Member name	Type	Description
charcollations- pecbufsize	unsigned short	Size of charcollationspecbuffer (at least 256+1).
charencoding- buffer	char *	Pointer to the char encoding string buffer.
charencoding- bufsize	unsigned short	Size of charencodingbuffer (at least 50+1).
checksum	a_bit_field	If set TRUE, global checksums are enabled (a checksum on every database page).
connectparms	const char *	Parameters needed to connect to the database.  They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".  The database server would be started by the connection string START parameter. For example: "START=c:\SQLA- ny12\bin32\dbeng12.exe".  A full example connection string including the START param- eter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLA- ny12\bin32\dbeng12.exe".
dbbufsize	unsigned short	Size of dbnamebuffer (for example, _MAX_PATH).
dbnamebuffer	char *	Pointer to the database file name buffer.
encrypted_ta- bles	a_bit_field	If set TRUE, encrypted tables are supported.
errortn	MSG_CALL- BACK	Address of an error message callback routine or NULL.
file_size	a_sql_uint32	Size of database file (in pages).
free_pages	a_sql_uint32	Number of free pages.
logbufsize	unsigned short	Size of lognamebuffer (for example, _MAX_PATH).
lognamebuffer	char *	Pointer to the transaction log file name buffer.
mirrorbufsize	unsigned short	Size of mirrornamebuffer (for example, _MAX_PATH).
mirrorname- buffer	char *	Pointer to the mirror file name buffer.

Member name	Type	Description
msgsrtn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
ncharcollations- pecbuffer	char *	Pointer to the nchar collation string buffer.
ncharcollations- pecbufsize	unsigned short	Size of ncharcollationspecbuffer (at least 256+1).
ncharencoding- buffer	char *	Pointer to the nchar encoding string buffer.
ncharencoding- bufsize	unsigned short	Size of ncharencodingbuffer (at least 50+1).
other_pages	a_sql_uint32	Number of pages that are not table pages, index pages, free pages, or bitmap pages.
page_usage	a_bit_field	Set TRUE to report page usage statistics, otherwise FALSE.  Set TRUE by dbinfo -u option.
quiet	a_bit_field	Set TRUE to operate without confirming messages.  Set TRUE by dbinfo -q option.
statusrtn	MSG_CALL- BACK	Address of a status message callback routine or NULL.
sysinfo	a_sysinfo	Inline a_sysinfo structure.
totals	a_table_info *	Pointer to a_table_info structure.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“a\\_table\\_info structure \[database tools\]” on page 898](#)
- [“a\\_sysinfo structure \[database tools\]” on page 897](#)
- [“DBInfo method \[database tools\]” on page 855](#)
- [“DBInfoDump method \[database tools\]” on page 856](#)
- [“DBInfoFree method \[database tools\]” on page 856](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)
- [“Corruption detection using checksums” \[SQL Anywhere Server - Database Administration\]](#)

## a\_db\_version\_info structure

Holds information regarding which version of SQL Anywhere was used to create the database.

### Syntax

```
typedef struct a_db_version_info
```

### Members

Member name	Type	Description
created_version	char	Set to one of VERSION_UNKNOWN, VERSION_PRE_10, etc.  indicating the server version that created the database file.
errortrn	MSG_CALLBACK	Address of an error message callback routine or NULL.
filename	const char *	Name of the database file to check.
msgtrn	MSG_CALLBACK	Address of an information message callback routine or NULL.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

### See also

- [“DBCcreatedVersion method \[database tools\]” on page 854](#)
- [“Version enumeration \[database tools\]” on page 867](#)

## a\_dblic\_info structure

Holds information containing licensing information.

### Syntax

```
typedef struct a_dblic_info
```

### Members

Member name	Type	Description
compname	char *	Company name for licensing.
conncount	a_sql_int32	Maximum number of connections licensed.  To set, use 1000000L for default.

Member name	Type	Description
errortrn	MSG_CALL- BACK	Address of an error message callback routine or NULL.
exename	char *	Name of the server executable or license file.
installkey	char *	Reserved; set NULL.  Set by dblic -k option.
msgtrn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
nodecount	a_sql_int32	Number of nodes licensed.
query_only	a_bit_field	Set TRUE to just display the license information.  Set FALSE to permit changing the information.
quiet	a_bit_field	Set TRUE to operate without printing messages.  Set TRUE by dblic -q option.
type	a_license_type	See lictype.h for values.  One of LICENSE_TYPE_PERSEAT, LICENSE_TYPE_CONCURRENT, or LICENSE_TYPE_PERCPU.
username	char *	User name for licensing.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**Remarks**

You must use this information only in a manner consistent with your license agreement.

**See also**

- [“DBLicense method \[database tools\]” on page 857](#)

**a\_dbtools\_info structure**

DBTools information callback used to initialize and finalize the DBTools library calls.

**Syntax**

```
typedef struct a_dbtools_info
```



**Members**

Member name	Type	Description
errortn	MSG_CALLBACK	Address of an error message callback routine or NULL.

**See also**

- [“DBToolsInit method \[database tools\]” on page 859](#)
- [“DBToolsFini method \[database tools\]” on page 858](#)

## a\_log\_file\_info structure

Used to obtain the log file and mirror log file information of a non-running database.

**Syntax**

```
typedef struct a_log_file_info
```

**Members**

Member name	Type	Description
dbname	const char *	Database file name.
encryption_key	const char *	The encryption key for the database file.
errortn	MSG_CALLBACK	Address of an error message callback routine or NULL.
logname	char *	Buffer for transaction log file name, or NULL.
logname_size	size_t	Size of buffer for transaction log file name, or zero.
mirrorname	char *	Buffer for mirror log file name, or NULL.
mirrorname_size	size_t	Size of buffer for mirror log file name, or zero.
reserved	void *	Reserved for internal use and must set to NULL.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“DBLogFileInfo method \[database tools\]” on page 857](#)

## a\_name structure

Specifies a variable list of names.

### Syntax

```
typedef struct a_name
```

### Members

Member name	Type	Description
name	char	One or more bytes comprising the name.
next	struct a_name *	Pointer to the next name in the list or NULL.

## a\_remote\_sql structure

Holds information needed for the dbremote utility using the DBTools library.

### Syntax

```
typedef struct a_remote_sql
```

### Members

Member name	Type	Description
apply	a_bit_field	Normally set TRUE.  When not set, messages are scanned but not applied. Corresponds to dbremote -a option.
argv	char **	Pointer to a parsed command line (a vector of pointers to strings).  If not NULL, then DBRemoteSQL will call a message routine to display each command line argument except those prefixed with -c, -cq, or -ek.
batch	a_bit_field	When set TRUE, force exit after applying message and scanning log (this is the same as at least one user having 'always' send time).  When cleared, allow run mode to be determined by remote users send times.

Member name	Type	Description
confirmrtn	MSG_CALLBACK	Address of a confirmation request callback routine or NULL.
connectparms	char *	<p>Parameters needed to connect to the database.</p> <p>They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".</p> <p>A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".</p>
debug	a_bit_field	When set TRUE, debug output is included.
debug_dump_size	a_sql_uint32	Reserved for internal use and must set to 0.
debug_page_offsets	a_bit_field	Reserved for internal use and must set to FALSE.
default_window_title	char *	A pointer to the default window title string.
deleted	a_bit_field	<p>Normally set TRUE.</p> <p>When not set, messages are not deleted after they are applied. Corresponds to dbremote -p option.</p>
encryption_key	char *	Pointer to an encryption key. Corresponds to the dbremote -ek option.
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
frequency	a_sql_uint32	<p>Set the polling frequency for incoming messages.</p> <p>This value should be set to the max( 1, receive_delay/ 60 ). See receive_delay below.</p>
full_q_scan	a_bit_field	Reserved for internal use and must set to FALSE.
include_scan_range	char *	Reserved for internal use and must set to NULL.

Member name	Type	Description
latest_backup	a_bit_field	When set TRUE, only logs that are backed up are processed.  Don't send operations from a live log. Corresponds to the dbremote -u option.
link_debug	a_bit_field	When set TRUE, debugging will be turned on for links.
locale	char *	Reserved for internal use and must set to NULL.
log_file_name	const char *	Pointer to the name of the DBRemoteSQL output log to which the message callbacks print their output.  If send is TRUE, the error log is sent to the consolidated (unless this pointer is NULL).
log_size	a_sql_uint32	DBRemoteSQL renames and restarts the online transaction log when the size of the online transaction log is greater than this value.  Corresponds to the dbremote -x option.
logrtn	MSG_CALLBACK	Pointer to a function that prints the given message to a log file.  These messages do not need to be seen by the user.
max_length	a_sql_uint32	Set to the maximum length (in bytes) a message can have.  This affects sending and receiving. The recommended value is 50000. Corresponds to the dbremote -l option.
memory	a_sql_uint32	Set to the maximum size (in bytes) of memory buffers to use while building messages to send.  The recommended value is at least 2 * 1024 * 1024. Corresponds to the dbremote -m option.
mirror_logs	char *	Pointer to the name of the directory containing offline mirror transaction logs.  Corresponds to the dbremote -ml option.
more	a_bit_field	This should be set to TRUE.

Member name	Type	Description
msgqueue rtn	MSG_QUEUE_CALLBACK	<p>Function called by DBRemoteSQL when it wants to sleep.</p> <p>The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h.</p> <ul style="list-style-type: none"> <li>• MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return.</li> <li>• MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.</li> </ul>
msg rtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
no_user_interaction	a_bit_field	When set TRUE, no user interaction is requested.
operations	a_sql_uint32	<p>This value is used when applying messages.</p> <p>Commits are ignored until DBRemoteSQL has at least this number of operations(inserts, deletes, updates) that are uncommitted. Corresponds to the dbremote -g option.</p>
patience_retry	a_sql_uint32	<p>Set this to the number of polls for incoming messages that DBRemoteSQL should wait before assuming that a message it is expecting is lost.</p> <p>For example, if patience_retry is 3 then DBRemoteSQL tries up to three times to receive the missing message. Afterward, it sends a resend request. The recommended value is 1. Corresponds to the dbremote -rp option.</p>
progress_index_rtn	SET_PROGRESS_CALLBACK	<p>Pointer to a function that updates the state of the progress bar.</p> <p>This function takes two unsigned integer arguments index and max. On the first call, the values are the minimum and maximum values (for example, 0, 100). On subsequent calls, the first argument is the current index value (for example, between 0 and 100) and the second argument is always 0.</p>

Member name	Type	Description
progress_msg_rtn	MSG_CALLBACK	Pointer to a function that displays a progress message.
queueparms	char *	Reserved for internal use and must set to NULL.
receive	a_bit_field	When set TRUE, messages are received.  If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -r option.
receive_delay	a_sql_uint32	Set this to the time (in seconds) to wait between polls for new incoming messages.  The recommended value is 60. Corresponds to the dbremote -rd option.
remote_output_file_name	char *	Pointer to the name of the DBRemoteSQL remote output file.  Corresponds to the dbremote -ro or -rt option.
rename_log	a_bit_field	When set TRUE, logs are renamed and restarted (DBRemoteSQL only).
resend_urgency	a_sql_uint32	Set the time (in seconds) that DBRemoteSQL waits after seeing that a user needs a rescan before performing a full scan of the log.  Set to zero to allow DBRemoteSQL to choose a good value based on user send times and other information it has collected. Corresponds to the dbremote -ru option.
scan_log	a_bit_field	Reserved for internal use and must set to FALSE.
send	a_bit_field	When set TRUE, messages are sent.  If receive and send are both FALSE then both are assumed TRUE. It is recommended to set receive and send FALSE. Corresponds to the dbremote -s option.
send_delay	a_sql_uint32	Set the time (in seconds) between scans of the log file for new operations to send.  Set to zero to allow DBRemoteSQL to choose a good value based on user send times. Corresponds to the dbremote -sd option.

Member name	Type	Description
set_window_title_rtn	SET_WINDOW_TITLE_CALLBACK	Pointer to a function that resets the title of the window (Windows only).  The title could be "database_name (receiving, scanning, or sending) - default_window_title".
threads	a_sql_uint32	Set the number of worker threads that should be used to apply messages.  This value must not exceed 50. Corresponds to the dbremote -w option.
transaction_logs	char *	Should identify the directory with offline transaction logs (DBRemoteSQL only).  Corresponds to the transaction_logs_directory argument of dbremote.
triggers	a_bit_field	This should usually be cleared (FALSE) in most cases.  When set TRUE, trigger actions are replicated. Care should be exercised.
truncate_remote_output_file	a_bit_field	When set TRUE, the remote output file is truncated rather than appended to.  Corresponds to the dbremote -rt option.
unused	a_bit_field	Reserved for internal use and must set to FALSE.
use_hex_offsets	a_bit_field	When set TRUE, log offsets are shown in hexadecimal notation; otherwise decimal notation is used.
use_relative_offsets	a_bit_field	When set TRUE, log offsets are displayed as relative to the start of the current log file.  When set FALSE, log offsets from the beginning of time are displayed.
verbose	a_bit_field	When set, extra information is produced.  Corresponds to the dbremote -v option.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

Member name	Type	Description
warningrtn	MSG_CALLBACK	Pointer to a function that displays the given warning message.  If NULL, the errorrtn function is called instead.

**Remarks**

The dbremote utility sets the following defaults before processing any command-line options:

- version = DB\_TOOLS\_VERSION\_NUMBER
- argv = (argument vector passed to application)
- deleted = TRUE
- apply = TRUE
- more = TRUE
- link\_debug = FALSE
- max\_length = 50000
- memory = 2 \* 1024 \* 1024
- frequency = 1
- threads = 0
- receive\_delay = 60
- send\_delay = 0
- log\_size = 0
- patience\_retry = 1
- resend\_urgency = 0
- log\_file\_name = (set from command line)
- truncate\_remote\_output\_file = FALSE
- remote\_output\_file\_name = NULL
- no\_user\_interaction = TRUE (if user interface is not available)
- errorrtn = (address of an appropriate routine)
- msggrtn = (address of an appropriate routine)



- confirmrtn = (address of an appropriate routine)
- msgqueuertn = (address of an appropriate routine)
- logrtn = (address of an appropriate routine)
- warningrtn = (address of an appropriate routine)
- set\_window\_title\_rtn = (address of an appropriate routine)
- progress\_msg\_rtn = (address of an appropriate routine)
- progress\_index\_rtn = (address of an appropriate routine)

**See also**

- [“DBRemoteSQL method \[database tools\]” on page 858](#)

## a\_sync\_db structure

Holds information needed for the dbmlsync utility using the DBTools library.

**Syntax**

```
typedef struct a_sync_db
```

**Members**

Member name	Type	Description
allow_outside_connect	a_bit_field	Reserved; use 0.
allow_schema_change	a_bit_field	Set TRUE to check for schema changes between synchronizations.  Equivalent to the dbmlsync -sc option.
apply_dnld_file	const char *	Name of download file to apply.  Equivalent to dbmlsync -ba option or NULL if option not specified.
argv	char **	The argv array for this run, the last element of the array must be NULL.
autoclose	a_bit_field	Set TRUE to close window on completion.  Equivalent to the dbmlsync -qc option.

Member name	Type	Description
background_retry	a_sql_int32	Number of times to retry an interrupted background synchronization.  Equivalent to the dbmsync -bkr option.
background_sync	a_bit_field	Set TRUE to do a background synchronization.  Equivalent to the dbmsync -bk option.
cache_verbosity	a_bit_field	Reserved; use 0.
ce_argv	char **	Reserved; use NULL.
ce_reproc_argv	char **	Reserved; use NULL.
changing_pwd	a_bit_field	Set TRUE when setting a new MobiLink password.  See new_mpassword field. Equivalent to the dbmsync -mn option.
confirmrtn	MSG_CALLBACK	Address of a confirmation request callback routine or NULL.
connectparms	char *	Parameters needed to connect to the database.  They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".  The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".  A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".
connectparms_allocated	a_bit_field	Reserved; use 0.
continue_download	a_bit_field	Set TRUE to continue a previously failed download.  Equivalent to the dbmsync -dc option.

Member name	Type	Description
create_dnld_file	const char *	Name of download file to create.  Equivalent to dbmlsync -bc option or NULL if switch not specified.
debug	a_bit_field	Reserved; use 0.
debug_dump_char	a_bit_field	Reserved; use 0.
debug_dump_hex	a_bit_field	Reserved; use 0.
debug_dump_size	a_sql_uint32	Reserved; use 0.
debug_page_offsets	a_bit_field	Reserved; use 0.
default_window_title	char *	Name of the program to display in the window caption (for example, DBMLSync).
dl_insert_width	a_sql_uint32	Reserved; use 0.
dl_use_put	a_bit_field	Reserved; use 0.
dlg_info_msg	a_sql_uint32	Reserved; use 0.
dnld_fail_len	a_sql_uint32	Reserved; use 0.
dnld_file_extra	const char *	Specify extra string to include in download file.  Equivalent to dbmlsync -be option.
dnld_gen_num	a_bit_field	Set TRUE to update generation number when download file is applied.  Equivalent to the dbmlsync -bg option.
dnld_read_size	a_sql_uint32	Set the download read size.  Equivalent to the dbmlsync -drs option.
download_only	a_bit_field	Set TRUE to perform download-only synchronization.  Equivalent to the dbmlsync -ds option.
encrypted_stream_opts	const char *	Reserved; use NULL.

Member name	Type	Description
encryption_key	char *	The encryption key for the database file. Equivalent to the dbmsync -ek option.
entered_dialog	a_bit_field	Reserved; use 0.
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
est_upld_row_cnt	a_sql_uint32	Set the estimated upload row count (for optimization). Equivalent to the dbmsync -urc option.
extended_options	char *	Extended options in the form "keyword=value;...". Equivalent to dbmsync -e option.
hide_conn_str	a_bit_field	Set FALSE to show connect string, TRUE to hide the connect string. Equivalent to the dbmsync -vc option.
hide_ml_pwd	a_bit_field	Set FALSE to show MobiLink password, TRUE to hide the MobiLink password. Equivalent to the dbmsync -vp option.
hovering_frequency	a_sql_uint32	Set the logscan polling period in seconds. Usually 60. Equivalent to the dbmsync -pp option.
ignore_debug_interrupt	a_bit_field	Reserved; use 0.
ignore_hook_errors	a_bit_field	Set TRUE to ignore errors that occur in hook functions. Equivalent to the dbmsync -eh option.
ignore_hovering	a_bit_field	Set TRUE to disable logscan polling. Equivalent to the dbmsync -p option.
ignore_scheduling	a_bit_field	Set TRUE to ignore scheduling. Equivalent to the dbmsync -is option.

Member name	Type	Description
include_scan_range	const char *	Reserved; use NULL.
init_cache	a_sql_uint32	Initial size for cache. Equivalent to the dbmsync -ci option.
init_cache_suffix	char	Suffix for initial cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).
kill_other_connections	a_bit_field	Set TRUE to drop connections with locks on tables being synchronized. Equivalent to the dbmsync -d option.
last_upload_def	a_syncpub *	Reserved; use NULL.
lite_blob_handling	a_bit_field	Reserved; use 0.
log_file_name	const char *	Database server message log file name. Equivalent to dbmsync -o or -ot option.
log_size	a_sql_uint32	Size in bytes of log file when renaming and restarting the transaction log. Specify 0 for unspecified size. Equivalent to the dbmsync -x option.
logrtn	MSG_CALLBACK	Address of a logging callback routine to write messages only to a log file or NULL.
max_cache	a_sql_uint32	Maximum size for cache. Equivalent to the dbmsync -cm option.
max_cache_suffix	char	Suffix for maximum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).
min_cache	a_sql_uint32	Minimum size for cache. Equivalent to the dbmsync -cl option.
min_cache_suffix	char	Suffix for minimum cache size ('B' for bytes, 'P' for percentage, or 0 if not specified).

Member name	Type	Description
mlpassword	char *	The MobiLink password or NULL, if the option is not specified.  Equivalent to the dbmsync -mp option.
msgqueuertn	MSG_QUEUE_CALLBACK	Function called by DBMLSync when it wants to sleep.  The parameter specifies the sleep period in milliseconds. The function should return the following, as defined in dllapi.h. <ul style="list-style-type: none"> <li>• MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. This is usually the value you should return.</li> <li>• MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.</li> <li>• MSGQ_SYNC_REQUESTED indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.</li> </ul>
msg rtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
new_mlpassword	char *	The new MobiLink password or NULL, if the option is not specified.  Equivalent to the dbmsync -mn option.
no_offline_logscan	a_sql_uint32	Set TRUE to disable offline logscan (cannot use with -x).  Equivalent to the dbmsync -do option.
no_schema_cache	a_bit_field	Reserved; use 0.
no_stream_compress	a_bit_field	Reserved; use 0.
offline_dir	const char *	Transaction logs directory.  Last item specified on dbmsync command line.

Member name	Type	Description
output_to_file	a_bit_field	Reserved; use 0.
output_to_mobile_link	a_bit_field	Reserved; use 1.
persist_connection	a_bit_field	Set TRUE to persist the MobiLink connection between synchronizations.  Set FALSE to close the MobiLink connection between synchronizations. Equivalent to the dbmsync -pc{+ -} option.
ping	a_bit_field	Set TRUE to ping MobiLink server.  Equivalent to the dbmsync -pi option.
preload_dlls	char *	Reserved; use NULL.
progress_index_rtn	SET_PROGRESS_CALLBACK	Function called to update the state of the progress bar.
progress_msg_rtn	MSG_CALLBACK	Function called to change the text in the status window, above the progress bar.
prompt_again	a_bit_field	Reserved; use 0.
prompt_for_encrypt_key	a_bit_field	Reserved; use 0.
raw_file	const char *	Reserved; use NULL.
rename_log	a_bit_field	Set TRUE to rename and restart the transaction log.  See log_size field. Equivalent to the dbmsync -x option.
retry_remote_ahead	a_bit_field	Set TRUE to resend upload using remote offset on progress mismatch when remote offset is greater than consolidated offset.  Equivalent to the dbmsync -ra option.
retry_remote_behind	a_bit_field	Set TRUE to resend upload using remote offset on progress mismatch.  when remote offset is less than consolidated offset. Equivalent to the dbmsync -r or -rb option.

Member name	Type	Description
script_full_path	char *	Deprecated; use NULL.
server_mode	a_bit_field	Set TRUE to run in server mode. Equivalent to the dbmsync -sm option.
server_port	a_sql_uint32	Set communication port when running in server mode. Equivalent to the dbmsync -po option.
set_window_title_rtn	SET_WINDOW_TITLE_CALLBACK	Function to call to change the title of the dbmsync window (Windows only).
status_rtn	STATUS_CALLBACK	Reserved; use NULL.
strictly_ignore_trigger_ops	a_bit_field	Reserved; use 0.
sync_opt	char *	Reserved; use NULL.
sync_params	char *	User authentication parameters. Equivalent to the dbmsync -ap option.
sync_profile	char *	Synchronization profile to execute. Equivalent to the dbmsync -sp option.
trans_upload	a_bit_field	Set TRUE to upload each database transaction separately. Equivalent to the dbmsync -tu option.
upld_fail_len	a_sql_uint32	Reserved; use 0.
upload_defs	a_syncpub *	Linked list of publications/subscriptions to synchronize.
upload_only	a_bit_field	Set TRUE to perform upload-only synchronization. Equivalent to the dbmsync -uo option.
usage_rtn	USAGE_CALLBACK	Reserved; use NULL.
use_fixed_cache	a_bit_field	Reserved; use 0.
use_hex_offsets	a_bit_field	Reserved; use 0.



Member name	Type	Description
use_relative_offsets	a_bit_field	Reserved; use 0.
used_dialog_allocation	a_bit_field	Reserved; use 0.
user_name	char *	The MobiLink user to synchronize (deprecated). Equivalent to the dbmsync -u option.
verbose	a_bit_field	Reserved; use 0.
verbose_download	a_bit_field	Reserved; use 0.
verbose_download_data	a_bit_field	Reserved; use 0.
verbose_hook	a_bit_field	Set TRUE to show hook script information. Equivalent to the dbmsync -vs option.
verbose_minimum	a_bit_field	Set TRUE to set verbosity at a minimum. Equivalent to the dbmsync -v option.
verbose_option_info	a_bit_field	Set TRUE to show command line and extended options. Equivalent to the dbmsync -vo option.
verbose_protocol	a_bit_field	Reserved; use 0.
verbose_row_cnts	a_bit_field	Set TRUE to show upload/download row counts. Equivalent to the dbmsync -vn option.
verbose_row_data	a_bit_field	Set TRUE to show upload/download row values. Equivalent to the dbmsync -vr option.
verbose_server	a_bit_field	Reserved; use 0.
verbose_upload	a_bit_field	Set TRUE to show upload stream information. Equivalent to the dbmsync -vu option.

Member name	Type	Description
verbose_up-load_data	a_bit_field	Reserved; use 0.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).
warningrtn	MSG_CALLBACK	Function called to display warning messages.

### Remarks

Some members correspond to features accessible from the dbmlsync command line utility. Unused members should be assigned the value 0, FALSE, or NULL, depending on data type.

### See also

- [“DBSynchronizeLog method \[database tools\]” on page 858](#)
- [“dbmlsync syntax” \[MobiLink - Client Administration\]](#)
- [“DBTools interface for dbmlsync” \[MobiLink - Client Administration\]](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## a\_syncpub structure

Holds information needed for the dbmlsync utility.

### Syntax

```
typedef struct a_syncpub
```

### Members

Member name	Type	Description
ext_opt	char *	Extended options in the form "keyword=value;...". These are the same options the would follow the dbmlsync -eu option.
next	struct a_syncpub *	Pointer to the next node in the list, NULL for the last node.
pub_name	char *	Publication name(s) separated by commas (deprecated). This is the same string that would follow the dbmlsync -n option. Only 1 of pub_name and subscription may be non-NULL.

Member name	Type	Description
subscription	char *	Subscription name(s) separated by commas.  This is the same string the would follow the dbmlsync -s option. Only 1 of pub_name and subscription may be non-NULL.

**See also**

- [“a\\_sync\\_db structure \[database tools\]” on page 887](#)
- [“DBSynchronizeLog method \[database tools\]” on page 858](#)
- [“DBTools interface for dbmlsync” \[MobiLink - Client Administration\]](#)

## a\_sysinfo structure

Holds information needed for dbinfo and dbunload utilities using the DBTools library.

**Syntax**

```
typedef struct a_sysinfo
```

**Members**

Member name	Type	Description
blank_padding	a_bit_field	1 if blank padding is used in this database, 0 otherwise.
case_sensitivity	a_bit_field	1 if the database is case sensitive, 0 otherwise.
default_collation	char	The collation sequence for the database.
encryption	a_bit_field	1 if the database is encrypted, 0 otherwise.
page_size	unsigned short	The page size for the database.
valid_data	a_bit_field	1 to indicate that the other bit fields are valid.

**See also**

- [“a\\_db\\_info structure \[database tools\]” on page 874](#)
- [“DBInfo method \[database tools\]” on page 855](#)
- [“DBInfoDump method \[database tools\]” on page 856](#)
- [“DBInfoFree method \[database tools\]” on page 856](#)

## a\_table\_info structure

Holds information about a table needed as part of the a\_db\_info structure.

### Syntax

```
typedef struct a_table_info
```

### Members

Member name	Type	Description
index_pages	a_sql_uint32	Number of index pages.
index_used	a_sql_uint32	Number of bytes used in index pages.
index_used_pct	a_sql_uint32	Index space utilization as a percentage.
next	struct a_table_info *	Next table in the list.
table_id	a_sql_uint32	ID number for this table.
table_name	char *	Name of the table.
table_pages	a_sql_uint32	Number of table pages.
table_used	a_sql_uint32	Number of bytes used in table pages.
table_used_pct	a_sql_uint32	Table space utilization as a percentage.

### See also

- [“a\\_db\\_info structure \[database tools\]” on page 874](#)
- [“DBInfo method \[database tools\]” on page 855](#)
- [“DBInfoDump method \[database tools\]” on page 856](#)
- [“DBInfoFree method \[database tools\]” on page 856](#)

## a\_translate\_log structure

Holds information needed for transaction log translation using the DBTools library.

### Syntax

```
typedef struct a_translate_log
```

**Members**

Member name	Type	Description
ansi_sql	a_bit_field	Set TRUE to produce ANSI standard SQL transactions.  Set TRUE by dbtran -s option.
chronological_order	a_bit_field	Reserved; set to FALSE.
comment_trigger_trans	a_bit_field	Set TRUE to include trigger-generated transactions as comments.  Set TRUE by dbtran -z option.
confirmrtn	MSG_CALLBACK	Address of a confirmation request callback routine or NULL.
connectparms	const char *	Parameters needed to connect to the database.  They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".  The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".  A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".
debug	a_bit_field	Reserved; set to FALSE.
debug_dump_char	a_bit_field	Reserved; set to FALSE.
debug_dump_hex	a_bit_field	Reserved; set to FALSE.
debug_dump_size	a_sql_uint32	Reserved, use 0.
debug_page_offsets	a_bit_field	Reserved; set to FALSE.
debug_sql_remote	a_bit_field	Reserved; set to FALSE.
encryption_key	const char *	The encryption key for the database file. Equivalent to dbtran -ek option.
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
extra_audit	a_bit_field	Reserved; set to FALSE.

Member name	Type	Description
force_chaining	a_bit_field	Reserved; set to FALSE.
force_recovery	a_bit_field	Reserved; set to FALSE.
generate_reciprocals	a_bit_field	Reserved; set to FALSE.
include_audit	a_bit_field	Reserved; set to FALSE.
include_destination_sets	const char *	Reserved, use NULL.
include_publications	const char *	Reserved, use NULL.
include_scan_range	const char *	Reserved, use NULL.
include_source_sets	const char *	Reserved, use NULL.
include_subsets	a_bit_field	Reserved; set to FALSE.
include_tables	const char *	Reserved, use NULL.
include_trigger_trans	a_bit_field	Set TRUE to include trigger-generated transactions. Set TRUE by dbtran -t, -g and -sr options.
leave_output_on_error	a_bit_field	Set TRUE to leave the generated SQL file if log error detected. Set TRUE by dbtran -k option.
logname	const char *	Name of the transaction log file. If NULL, there is no log.
logrtn	MSG_CALLBACK	Address of a logging callback routine to write messages only to a log file or NULL.
logs_dir	const char *	Transaction logs directory. Equivalent to dbtran -m option. The sqlname pointer must be set and connectparms must be NULL.
match_mode	a_bit_field	Reserved; set to FALSE.
match_pos	const char *	Reserved, use NULL.

Member name	Type	Description
msgsrtn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
omit_comments	a_bit_field	Reserved; set to FALSE.
queueparms	const char *	Reserved, use NULL.
quiet	a_bit_field	Set to TRUE to operate without printing messages.  Set TRUE by dbtran -q option.
recovery_bytes	a_sql_uint32	Reserved, use 0.
recovery_ops	a_sql_uint32	Reserved, use 0.
remove_rollback	a_bit_field	Set to FALSE if you want to include rollback transactions in output.  Set FALSE by dbtran -a option.
replace	a_bit_field	Set TRUE to replace the SQL file without a confirmation.  Set TRUE by dbtran -y option.
repsrver_users	const char *	Reserved, use NULL.
show_undo	a_bit_field	Reserved; set to FALSE.
since_checkpoint	a_bit_field	Set TRUE for output from most recent checkpoint.  Set TRUE by dbtran -f option.
since_time	a_sql_uint32	Output from most recent checkpoint before time.  The number of minutes since January 1, 0001. Equivalent to dbtran -j option.
sqlname	const char *	Name of the SQL output file.  If NULL, then the name is based on the transaction log file name. Equivalent to dbtran -n option.
statusrtn	MSG_CALL- BACK	Address of a status message callback routine or NULL.
use_hex_offsets	a_bit_field	Reserved; set to FALSE.
use_relative_offsets	a_bit_field	Reserved; set to FALSE.

Member name	Type	Description
userlist	p_name	A linked list of user names.  Equivalent to dbtran -u user1,... or -x user1,... Select or omit transactions for listed users.
userlisttype	char	Set to DBTRAN_INCLUDE_ALL unless you want to include or exclude a list of users.  DBTRAN_INCLUDE_SOME for -u, or DBTRAN_EXCLUDE_SOME for -x.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“a\\_name structure \[database tools\]” on page 880](#)
- [“UserList enumeration \[database tools\]” on page 865](#)
- [“DBTranslateLog method \[database tools\]” on page 860](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## a\_truncate\_log structure

Holds information needed for transaction log truncation using the DBTools library.

**Syntax**

```
typedef struct a_truncate_log
```

**Members**

Member name	Type	Description
connect-params	const char *	Parameters needed to connect to the database.  They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".  The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".  A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".



Member name	Type	Description
errorrtn	MSG_CALL- BACK	Address of an error message callback routine or NULL.
msgsrtn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
quiet	a_bit_field	Set TRUE to operate without printing messages.  Set TRUE by dbbackup -q option.
serv- er_backup	a_bit_field	Set TRUE to indicate backup on server using BACKUP DATABASE.  Set TRUE by dbbackup -s option when dbbackup -x option is specified.
trun- cate_inter- rupted	char	Truncate was interrupted if non-zero.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“DBTruncateLog method \[database tools\]” on page 860](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

**a\_validate\_db structure**

Holds information needed for database validation using the DBTools library.

**Syntax**

```
typedef struct a_validate_db
```

**Members**

Member name	Type	Description
connect-params	const char *	<p>Parameters needed to connect to the database.</p> <p>They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".</p> <p>A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".</p>
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
index	a_bit_field	<p>Set TRUE to validate indexes.</p> <p>The tables field points to a list of indexes. Set TRUE by dbvalid -i option. Set FALSE by dbvalid -t option.</p>
msgsrtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
quiet	a_bit_field	<p>Set TRUE to operate without printing messages.</p> <p>Set TRUE by dbvalid -q option.</p>
statusrtn	MSG_CALLBACK	Address of a status message callback routine or NULL.
tables	p_name	<p>Pointer to a linked list of table names or index names (when the index field is set TRUE).</p> <p>This is set by the dbvalid object-name-list argument.</p>
type	char	<p>The type of validation to perform.</p> <p>One of VALIDATE_NORMAL, VALIDATE_EXPRESS, VALIDATE_CHECKSUM, etc. See Validation enumeration.</p>
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“a\\_name structure \[database tools\]” on page 880](#)
- [“DBValidate method \[database tools\]” on page 862](#)
- [“Validation enumeration \[database tools\]” on page 865](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

## an\_erase\_db structure

Holds information needed to erase a database using the DBTools library.

**Syntax**

```
typedef struct an_erase_db
```

**Members**

Member name	Type	Description
confirmrtn	MSG_CALLBACK	Address of a confirmation request callback routine or NULL.
dbname	const char *	Database file name.
encryption_key	const char *	The encryption key for the database file. Equivalent to dberase -ek or -ep options.
erase	a_bit_field	Erase without confirmation (1) or with confirmation (0). Set TRUE by dberase -y option.
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
msgrtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
quiet	a_bit_field	Operate without printing messages (1), or print messages (0). Set TRUE by dberase -q option.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“DBErase method \[database tools\]” on page 855](#)
- [“Callback functions” on page 847](#)

## an\_unload\_db structure

Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote.

### Syntax

```
typedef struct an_unload_db
```

### Members

Member name	Type	Description
compress_output	a_bit_field	Set TRUE to compress table data files.  Set TRUE by dbunload -cp option.
confirmrtn	MSG_CALL- BACK	Address of a confirmation request callback routine or NULL.
connectparms	const char *	Parameters needed to connect to the database.  They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".  The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".  A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".
debug	a_bit_field	Reserved; set FALSE.
display_create	a_bit_field	Set TRUE to display database creation command (sql or dbinit).  Set TRUE by dbunload -cm sql or -cm dbinit option.
display_create_dbinit	a_bit_field	Set TRUE to display dbinit database creation command.  Set TRUE by dbunload -cm dbinit option.
encrypted_tables	a_bit_field	Set TRUE to enable encrypted tables in new database (with -an or -ar).  Set TRUE by dbunload/dbxtract -et option.

Member name	Type	Description
encryption_algorithm	const char *	The encryption algorithm which may be "simple", "aes", "aes256", "aes_fips", "aes256_fips", or NULL for none.  Set by dbunload/dbxtract -ea option.
encryption_key	const char *	The encryption key for the database file.  Set by dbunload/dbxtract -ek or -ep option.
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
escape_char	char	The escape character (normally, "\").  Used when escape_char_present is TRUE. Set TRUE by dbunload/dbxtract -p option.
escape_char_present	a_bit_field	Set TRUE to indicate that the escape character in escape_char is defined.  Set TRUE by dbunload/dbxtract -p option.
exclude_foreign_keys	a_bit_field	Set TRUE to exclude foreign keys.  Set TRUE by dbxtract -xf option.
exclude_hooks	a_bit_field	Set TRUE to exclude procedure hooks.  Set TRUE by dbxtract -xh option.
exclude_procedures	a_bit_field	Set TRUE to exclude stored procedures.  Set TRUE by dbxtract -xp option.
exclude_tables	a_bit_field	Set FALSE to indicate that the list contains tables to be included.  Set TRUE to indicate that the list contains tables to be excluded. Set TRUE by dbunload -e option.
exclude_triggers	a_bit_field	Set TRUE to exclude triggers.  Set TRUE by dbxtract -xt option.
exclude_views	a_bit_field	Set TRUE to exclude views.  Set TRUE by dbxtract -xv option.

Member name	Type	Description
extract	a_bit_field	Set TRUE if performing a remote database extraction. Set FALSE by dbunload. Set TRUE by dbxtract.
genscript	a_bit_field	Reserved; set FALSE.
in- clude_where_sub- scribe	a_bit_field	Set TRUE to extract fully qualified publications. Set TRUE by dbxtract -f option.
isolation_level	unsigned short	The isolation level at which to operate. Set by dbxtract -l option.
isolation_set	a_bit_field	Set TRUE to indicate that isolation_level has been set for all extraction operations. Set TRUE by dbxtract -l option.
locale	const char *	Reserved; use NULL.
make_auxiliary	a_bit_field	Set TRUE to make auxiliary catalog (for use with diagnostic tracing). Set TRUE by dbunload -k option.
ms_filename	const char *	Reserved; use NULL.
ms_reserve	int	Reserved; use 0.
ms_size	int	Reserved; use 0.
msgtrn	MSG_CALL- BACK	Address of an information message callback routine or NULL.
no_confirm	a_bit_field	Set TRUE to replace an existing SQL script file without confirmation. Set by dbunload/dbxtract -y option.
no_reload_status	a_bit_field	Set TRUE to suppress reload status messages for tables and indexes. Set TRUE by dbunload -qr option.
notemp_size	long	Reserved; use 0.

Member name	Type	Description
preserve_identity_values	a_bit_field	Set TRUE to preserve identity values for AUTOINCREMENT columns.  Set TRUE by dbunload -l option.
preserve_ids	a_bit_field	Set TRUE to preserve user IDs.  This is the normal setting. Set FALSE by dbunload -e option.
profiling_uses_single_dbspace	a_bit_field	Set TRUE to collapse to a single dbspace file (for use with diagnostic tracing).  Set TRUE by dbunload -kd option.
recompute	a_bit_field	Set TRUE to redo computed columns.  Set TRUE by dbunload -dc option.
refresh_mat_view	a_bit_field	Set TRUE to generate statements to refresh text indexes and valid materialized views.  Set TRUE by dbunload/dbxtract -g option.
reload_connect_parms	char *	Connection parameters such as user ID, password, and database for the reload database.  Set by dbunload/dbxtract -ac option.
reload_db_filename	char *	Name of the new database file to create and reload.  Set by dbunload/dbxtract -an option.
reload_db_logname	char *	Filename of the new database transaction log or NULL.  Set by dbxtract -al option.
reload_filename	const char *	Name to use for the reload SQL script file (for example, reload.sql).  Set by dbunload -r option.
reload_page_size	unsigned short	The reloaded database page size.  Set by dbunload -ap option.
remote_dir	const char *	Like temp_dir but for internal unloads on server side.

Member name	Type	Description
remove_encrypted_tables	a_bit_field	Set TRUE to remove encryption from encrypted tables. Set TRUE by dbunload/dbxtract -er option.
replace_db	a_bit_field	Set TRUE to replace the database. Set TRUE by dbunload -ar option.
runscript	a_bit_field	Reserved; set FALSE.
schema_reload	a_bit_field	Reserved; set FALSE.
site_name	const char *	The site name to be used by dbxtract. NULL otherwise.
start_subscriptions	a_bit_field	Set TRUE to start subscriptions. This is the default for dbxtract. Set FALSE by dbxtract -b option.
startline	const char *	Reserved; use NULL.
startline_name	a_bit_field	Reserved; set FALSE.
startline_old	const char *	Reserved; use NULL.
statusrtn	MSG_CALLBACK	Address of a status message callback routine or NULL.
subscriber_username	const char *	The subscriber name to be used by dbxtract. NULL otherwise.
sysinfo	a_sysinfo	Reserved; use NULL.
table_list	p_name	Selective table list. Set by dbunload -e and -t options.
table_list_provided	a_bit_field	Set TRUE to indicate that a list of tables has been provided. See table_list field. Set TRUE by dbunload -e or -t options.
temp_dir	const char *	Directory for unloading data files.
template_name	const char *	The template name to be used by dbxtract. NULL otherwise.
unload_interrupted	char	Reserved; set to 0.



Member name	Type	Description
unload_type	char	Set Unload enumeration (UNLOAD_ALL and so on). Set by dbunload/dbxtract -d, -k, -n options.
unordered	a_bit_field	Set TRUE for unordered data.  Indexes will not be used to unload data. Set by dbunload/dbxtract -u option.
use_internal_reload	a_bit_field	Set TRUE to perform an internal reload.  This is the normal setting. Set TRUE by dbunload/dbxtract -ii and -xi option. Set FALSE by dbunload/dbxtract -ix and -xx option.
use_internal_unload	a_bit_field	Set TRUE to Perform an internal unload.  Set TRUE by dbunload/dbxtract -i? option. Set FALSE by dbunload/dbxtract -x? option.
verbose	char	See Verbosity enumeration (VB_QUIET, VB_NORMAL, VB_VERBOSE).
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**Remarks**

Those fields used by the dbxtract SQL Remote Extraction utility are indicated.

**See also**

- [“Verbosity enumeration \[database tools\]” on page 866](#)
- [“DBUnload method \[database tools\]” on page 861](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#)

**an\_upgrade\_db structure**

Holds information needed to upgrade a database using the DBTools library.

**Syntax**

```
typedef struct an_upgrade_db
```

**Members**

Member name	Type	Description
connect-params	const char *	<p>Parameters needed to connect to the database.</p> <p>They take the form of connection strings, such as the following: "UID=DBA;PWD=sql;DBF=demo.db".</p> <p>The database server would be started by the connection string START parameter. For example: "START=c:\SQLAny12\bin32\dbeng12.exe".</p> <p>A full example connection string including the START parameter: "UID=DBA;PWD=sql;DBF=demo.db;START=c:\SQLAny12\bin32\dbeng12.exe".</p>
errorrtn	MSG_CALLBACK	Address of an error message callback routine or NULL.
jconnect	a_bit_field	<p>Set TRUE to upgrade the database to include jConnect procedures.</p> <p>Set FALSE by dbupgrad -i option.</p>
msgsrtn	MSG_CALLBACK	Address of an information message callback routine or NULL.
quiet	a_bit_field	<p>Set TRUE to operate without printing messages.</p> <p>Set TRUE by dbupgrad -q option.</p>
statusrtn	MSG_CALLBACK	Address of a status message callback routine or NULL.
version	unsigned short	DBTools version number (DB_TOOLS_VERSION_NUMBER).

**See also**

- [“DBUpgrade method \[database tools\]” on page 861](#)
- [“Callback functions” on page 847](#)
- [“Connection parameters” \[\*SQL Anywhere Server - Database Administration\*\]](#)

## Software component exit codes

All database tools library entry points use the following exit codes. The SQL Anywhere utilities (dbbackup, dbspawn, dbeng12, and so on) also use these exit codes.

Code	Status	Explanation
0	EXIT_OKAY	Success
1	EXIT_FAIL	General failure
2	EXIT_BAD_DATA	Invalid file format
3	EXIT_FILE_ERROR	File not found, unable to open
4	EXIT_OUT_OF_MEMORY	Out of memory
5	EXIT_BREAK	Terminated by the user
6	EXIT_COMMUNICATIONS_FAIL	Failed communications
7	EXIT_MISSING_DATABASE	Missing a required database name
8	EXIT_PROTOCOL_MISMATCH	Client/server protocol mismatch
9	EXIT_UNABLE_TO_CONNECT	Unable to connect to the database server
10	EXIT_ENGINE_NOT_RUNNING	Database server not running
11	EXIT_SERVER_NOT_FOUND	Database server not found
12	EXIT_BAD_ENCRYPT_KEY	Missing or bad encryption key
13	EXIT_DB_VER_NEWER	Server must be upgraded to run database
14	EXIT_FILE_INVALID_DB	File is not a database
15	EXIT_LOG_FILE_ERROR	Log file was missing or other error
16	EXIT_FILE_IN_USE	File in use
17	EXIT_FATAL_ERROR	Fatal error occurred
18	EXIT_MISSING_LICENSE_FILE	Missing server license file
19	EXIT_BACKGROUND_SYNC_ABORTED	Background synchronization aborted to allow higher priority operations proceed
20	EXIT_FILE_ACCESS_DENIED	Database cannot be started because access is denied
255	EXIT_USAGE	Invalid parameters on the command line

These exit codes are defined in the `%SQLANY12%\sdk\include\sqldef.h` file.

---

---

# Database and application deployment

When you have completed a database application, you must deploy the application to your end users. Depending on the way in which your application uses SQL Anywhere (as an embedded database, in a client/server fashion, and so on) you may have to deploy components of the SQL Anywhere software along with your application. You may also have to deploy configuration information, such as data source names, that enable your application to communicate with SQL Anywhere.

## Note

Redistribution of files is subject to your license agreement with Sybase. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

The following deployment steps are examined in this section:

- Determining required files based on the choice of application platform and architecture.
- Configuring client applications.

Much of the section deals with individual files and where they need to be placed. However, the recommended way of deploying SQL Anywhere components is to use the **Deployment Wizard** or to use a silent install. For information, see [“The Deployment Wizard” on page 919](#) and [“Silent installs using the SQL Anywhere installer” on page 923](#).

## Types of deployment

The files you need to deploy depend on the type of deployment you choose. Here are some possible deployment models:

- **Client deployment** You may deploy only the client portions of SQL Anywhere to your end users, so that they can connect to a centrally located network database server.
- **Network server deployment** You may deploy network servers to offices, and then deploy clients to each of the users within those offices.
- **Embedded database deployment** You may deploy an application that runs with the personal database server. In this case, both client and personal server need to be installed on the end-user's computer.
- **SQL Remote deployment** Deploying a SQL Remote application is an extension of the embedded database deployment model.
- **MobiLink deployment** For information about deploying MobiLink servers, see [“MobiLink application deployment” \[MobiLink - Server Administration\]](#).

- **Administration tools deployment** You may deploy Interactive SQL, Sybase Central and other management tools.

## Ways to distribute files

There are two ways to deploy SQL Anywhere:

- **Use the SQL Anywhere installer** You can make the installer available to your end users. By selecting the proper option, each end user is guaranteed to receive the files they need.

This is the simplest solution for many deployment cases. In this case, you must still provide your end users with a method for connecting to the database server (such as an ODBC data source).

For more information, see [“The Deployment Wizard” on page 919](#) or [“Silent installs using the SQL Anywhere installer” on page 923](#).

- **Develop your own installation** There may be reasons for you to develop your own installation program that includes SQL Anywhere files. This is a more complicated option, and most of this section addresses the needs of those who are developing their own installation.

If SQL Anywhere has already been installed for the server type and operating system required by the client application architecture, the required files can be found in the appropriately-named subdirectory, located in the SQL Anywhere installation directory. For example, the *bin32* subdirectory of your installation directory contains the files required to run the server for 32-bit Windows operating systems.

Whichever option you choose, you must not violate the terms of your license agreement.

## Installation directories and file names

For a deployed application to work properly, the database server and client applications must each be able to locate the files they need. The deployed files should be located relative to each other in the same fashion as your SQL Anywhere installation.

In practice, this means that on Windows, most files belong in a single directory. For example, on Windows both client and database server required files are installed in a single directory, which is the *bin32* subdirectory of the SQL Anywhere installation directory.

For information about the places where the software looks for files, see [“How SQL Anywhere locates files” \[SQL Anywhere Server - Database Administration\]](#).

## Linux, Unix, and Mac OS X deployment issues

Unix deployments are different from Windows deployments in some ways:

- **Directory structure** For Linux, Unix, and Mac OS X installations, the default directory structure is as follows:

Directory	Contents
<i>/opt/sqlanywhere12/bin32</i> and <i>/opt/sqlanywhere12/bin64</i>	Executable files, License files
<i>/opt/sqlanywhere12/lib32</i> and <i>/opt/sqlanywhere12/lib64</i>	Shared objects and libraries
<i>/opt/sqlanywhere12/res</i>	String files

On AIX, the default root directory is */usr/lpp/sqlanywhere12* instead of */opt/sqlanywhere12*.

On Mac OS X, the default root directory is */Applications/SQLAnywhere12/System* instead of */opt/sqlanywhere12*.

Depending on the complexity of your deployment, you might choose to put all of the files that you require for your application into a single directory. You may find that this is a simpler deployment option, especially if a small number of files are required for your deployment. This directory could be the same directory that you use for your own application.

- **File suffixes** In the tables in this section, the shared objects are listed with a suffix of *.so* or *.so.1*. The version number, 1, could be higher as updates are released. For simplicity, the version number is often not listed.

For AIX, the suffix does not contain a version number so it is simply *.so*.

- **Symbolic links** Each shared object is installed as a symbolic link (symlink) to a file of the same name with the additional suffix *.1* (one). For example, *libdblib12.so* is a symbolic link to the file *libdblib12.so.1* in the same directory.

The version suffix *.1* could be higher as updates are released and the symbolic link must be redirected.

On Mac OS X, you should create a jnilib symbolic link for any dylib that you want to load directly from your Java client application.

- **Threaded and non-threaded applications** Most shared objects are provided in two forms, one of which has the additional characters *\_r* before the file suffix. For example, in addition to *libdblib12.so.1*, there is a file named *libdblib12\_r.so.1*. In this case, threaded applications must be linked to the shared object whose name has the *\_r* suffix, while non-threaded applications must be linked to the shared object whose name does not have the *\_r* suffix. Occasionally, there is a third form of shared object with *\_n* before the file suffix. This is a version of the shared object that is used with non-threaded applications.

- **Character set conversion** If you want to use database server character set conversion, you need to include the following files:
  - *libdbicu12.so.1*
  - *libdbicu12\_r.so.1*
  - *libdbicudt12.so.1*
  - *sqlany.cvf*
- **Environment variables** On Linux, Unix, and Mac OS X, environment variables must be set for the system to be able to locate SQL Anywhere applications and libraries. It is recommended that you use the appropriate file for your shell, either *sa\_config.sh* or *sa\_config.csh* (located in the directories */opt/sqlanywhere12/bin32* and */opt/sqlanywhere12/bin64*) as a template for setting the required environment variables. Some of the environment variables set by these files include PATH, LD\_LIBRARY\_PATH, and SQLANY12.

For information about how SQL Anywhere looks for files, see [“How SQL Anywhere locates files” \[SQL Anywhere Server - Database Administration\]](#).

## File naming conventions

SQL Anywhere uses consistent file naming conventions to help identify and group system components.

These conventions include:

- **Version number** The SQL Anywhere version number is indicated in the file name of the main server components (executable files, dynamic link libraries, shared objects, license files, and so on).

For example, the file *dbeng12.exe* is a version 12 executable for Windows.

- **Language** The language used in a language resource library is indicated by a two-letter code within its file name. The two characters before the version number indicate the language used in the library. For example, *dblgen12.dll* is the message resource library for the English language. These two-letter codes are specified by ISO standard 639-1.

For more information about language labels, see [“Language Selection utility \(dblang\)” \[SQL Anywhere Server - Database Administration\]](#).

For a list of the languages available in SQL Anywhere, see [“Localized versions of SQL Anywhere” \[SQL Anywhere Server - Database Administration\]](#).

### Identifying other file types

The following table identifies the platform and function of SQL Anywhere files according to their file extension. SQL Anywhere follows standard file extension conventions where possible.

File extension	Platform	File type
<i>.bat, .cmd</i>	Windows	Batch command files



File extension	Platform	File type
<i>.chm, .chw</i>	Windows	Help system file
<i>.dll</i>	Windows	Dynamic Link Library
<i>.exe</i>	Windows	Executable file
<i>.ini</i>	All	Initialization file
<i>.lic</i>	All	License file
<i>.lib</i>	Varies by development tool	Static runtime libraries for the creation of embedded SQL executables
<i>.res</i>	Linux, Unix, Mac OS X	Language resource file for non-Windows environments
<i>.so</i>	Linux, Unix	Shared object or shared library file (the equivalent of a Windows DLL)
<i>.bundle, .dylib</i>	Mac OS X	Shared object file (the equivalent of a Windows DLL)

### Database file names

SQL Anywhere databases are composed of two elements:

- **Database file** This is used to store information in an organized format. By default, this file uses a *.db* file extension. There may also be additional dbspace files. These files could have any file extension including none.
- **Transaction log file** This is used to record all changes made to data stored in the database file. By default, this file uses a *.log* file extension, and is generated by SQL Anywhere if no such file exists and a log file is specified to be used. A transaction log mirror has the default extension of *.mlg*.

These files are updated, maintained and managed by the SQL Anywhere relational database management system.

## The Deployment Wizard

The SQL Anywhere **Deployment Wizard** is the preferred tool for creating deployments of SQL Anywhere for Windows. The **Deployment Wizard** can create installer files that include some or all the following components:

- Client interfaces, such as ODBC

- SQL Anywhere server, including remote data access, database tools, and encryption
- UltraLite relational database
- MobiLink server, client, and encryption
- QAnywhere messaging
- Administration tools such as Interactive SQL and Sybase Central

You can use the **Deployment Wizard** to create a Microsoft Windows Installer Package file or a Microsoft Windows Installer Merge Module file:

- **Microsoft Windows Installer Package file** A file containing the instructions and data required to install an application. An Installer Package file has the extension *.msi*.
- **Microsoft Windows Installer Merge Module file** A simplified type of Microsoft Installer Package file that includes all files, resources, registry entries, and setup logic to install a shared component. A merge module has the extension *.msm*.

A merge module cannot be installed alone because it lacks some vital database tables that are present in an installer package file. Merge modules contain additional tables that are unique to themselves. To install the information delivered by a merge module with an application, the module must first be merged into the application's Installer Package (*.msi*) file. A merge module consists of the following parts:

- A merge module database containing the installation properties and setup logic being delivered by the merge module.
- A merge module Summary Information Stream describing the module.
- A *MergeModule.CAB* cabinet file stored as a stream inside the merge module. This cabinet file contains all the files required by the components delivered by the merge module. Every file delivered by the merge module must be stored inside of a cabinet file that is embedded as a stream in the merge module's structured storage. In a standard merge module, the name of this cabinet is always *MergeModule.CAB*.

The **Deployment Wizard** allows you to select subsets of the components included in SQL Anywhere. Each component has dependencies on other components so the files that are selected by the wizard may include files from other categories.

If you want to determine what files are included in each selectable component, create an MSI installer image and select all the components. A log file is created that specifies what files are included in every component. This text file can be examined with a text editor. There are headings such as **Feature: SERVER32\_TOOLS** and **Feature: CLIENT64\_TOOLS** that closely correspond to the **Deployment Wizard** components. Looking at the file gives you an idea of what is included in each group.

To run the **Deployment Wizard**, click **Start » Programs » SQL Anywhere 12 » Administration Tools » Deploy to Windows**.

Make sure to record the product code that was used to create the deployment package. You can use the product code at a later time to uninstall the package.

**Note**

Redistribution of files is subject to your license agreement. You must acknowledge that you are properly licensed to redistribute SQL Anywhere files. Check your license agreement before proceeding.

**See also**

- [“Running the Deployment Wizard” on page 921](#)
- [“Deployment package installation utility” on page 922](#)
- [“Silent installs using the SQL Anywhere installer” on page 923](#)

## Running the Deployment Wizard

The SQL Anywhere **Deployment Wizard** is used to create database and application deployments for Windows.

**Prerequisites**

Windows operating system

**Context and remarks**

Database and application deployment

**Create a deployment file**

1. To run the **Deployment Wizard**, click **Start » Programs » SQL Anywhere 12 » Administration Tools » Deploy to Windows**.
2. Follow the instructions in the wizard.
3. Record the product code that was used to create the deployment package. An example of the product code is {7C99D24E-AE1B-4770-9015-65B805950E3D}. You can use the product code at a later time to uninstall the package.

**Results**

The **Destination Path** that you selected will contain the deployment package (for example, *sqlany12.msi*) and a log file (*sqlany12.msi.log*).

**Next**

Once you have created a deployment package, you should test it by installing it.

## Example

You can also run the **Deployment Wizard** from the *Deployment* subdirectory of your SQL Anywhere installation. For example:

```
%SQLANY12%\Deployment\DeploymentWizard.exe
```

## See also

- “The Deployment Wizard” on page 919
- “Deployment package installation utility” on page 922

## Deployment package installation utility

Installs or uninstalls deployments on Windows.

### Syntax

```
msiexec [ options ] [ SQLANYDIR=path ]
```

Option	Description
<i>/log log-filename</i>	This parameter tells the Microsoft Windows Installer to log operations to a file (for example, <i>sqlany12.log</i> ). The log file is useful for detecting problems.
<i>/package package-name</i>	This parameter tells the Microsoft Windows Installer to install the specified package (for example, <i>sqlany12.msi</i> ).
<i>/qn</i>	This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.
<i>/uninstall package-name   product-code</i>	This parameter tells the Microsoft Windows Installer to uninstall the product associated with the specified MSI file or product code. The <i>product-code</i> is created when the <b>Deployment Wizard</b> is run to create the deployment package.
<b>SQLANYDIR=path</b>	The value of this parameter sets the default path to where the software is to be installed. This option is useful for silent installs. It is ignored for <b>/uninstall</b> .

## Remarks

The `/qn` option can be used to perform the operation without user messages. On Windows Vista and later versions of Windows, administrator privilege is required to install or uninstall software. When the `/qn` option is used, the automatic request for administrator privilege is suppressed.

## See also

- [“The Deployment Wizard” on page 919](#)
- [“Running the Deployment Wizard” on page 921](#)

## Example

The following command installs a deployment file.

```
msiexec /package sqlany12.msi
```

The following command performs a silent install of the deployment file to the specified folder and logs the results to a file.

```
msiexec /qn /package sqlany12.msi /log sqlany12.log SQLANYDIR=c:\sa12
```

The following command performs a silent uninstall using the specified package and logs the results to a file.

```
msiexec /qn /uninstall sqlany12.msi /log sqlany12.log
```

The following command performs a silent uninstall using the specified product code.

```
msiexec.exe /qn /uninstall {19972A31-72EF-126F-31C7-5CF249B8593F}
```

# Silent installs using the SQL Anywhere installer

Silent installs run without user input and with no indication to the user that an install is occurring. On Windows operating systems, you can call the SQL Anywhere installer from your own install program in such a way that the SQL Anywhere install is silent.

The common options for the SQL Anywhere install program *setup.exe* are:

- **/L:language\_id** The language identifier is a locale number that represents the language for the install. For example, locale ID 1033 identifies U.S. English, locale ID 1031 identifies German, locale ID 1036 identifies French, locale ID 1041 identifies Japanese, and locale ID 2052 identifies Simplified Chinese.
- **/S** This option hides the initialization window. Use this option in conjunction with **/V**.
- **/V** Specify parameters to MSIEXEC, the Microsoft Windows Installer tool.

The following command line example assumes that the install image directory is in the *software* \SQLAnywhere directory on the disk in drive *d:*.

```
d:\software\sqlanywhere\setup.exe /l:1033 /s "/v: /qn
REGKEY=QEDEV-B888A-6L123-45678-90123 INSTALLDIR=c:\sa12 DIR_SAMPLES=c:
\sa12\Samples"
```

**Note**

The setup.exe in the command above is the one located in the same directory as the SQLANY32.msi and SQLANY64.msi files. The setup.exe in the parent directory of those files does NOT support silent installs.

The following properties apply to the SQL Anywhere installer:

- **INSTALLDIR** The value of this parameter is the path to where the software is installed.
- **DIR\_SAMPLES** The value of this parameter is the path to where the sample programs are installed.
- **DIR\_SQLANY\_MONITOR** The value of this parameter is the path to where the SQL Anywhere Monitor database (*samonitor.db*) is installed.
- **USERNAME** The value of this parameter is the user name to record for this installation (for example, USERNAME="John Smith").
- **COMPANYNAME** The value of this parameter is the company name to record for this installation (for example, COMPANYNAME="Smith Holdings").
- **REGKEY** The value of this parameter must be a valid software registration key.
- **REGKEY\_ADD\_1** The value of this parameter must be a valid software registration key for an add-on feature such as ECC or FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.
- **REGKEY\_ADD\_2** The value of this parameter must be a valid software registration key for an add-on feature such as ECC or FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.
- **REGKEY\_ADD\_3** The value of this parameter must be a valid software registration key for an add-on feature such as ECC or FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

The following example shows how to specify the SQL Anywhere install properties:

```
d:\software\sqlanywhere\setup.exe /S "/v: /qn
USERNAME="John Smith"
COMPANYNAME="Smith Holdings"
REGKEY=QEDEV-B888A-6L123-45678-90123
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sa12
DIR_SAMPLES=c:\sa12\Samples
DIR_SQLANY_MONITOR=c:\sa12\Monitor"
```

Although the above text is shown over several lines for reasons of length, it would be specified as a single line of text. Note the use of the backslash character to escape the interior quotation marks.

The following properties apply to the SQL Anywhere Monitor installer:

- **INSTALLDIR** The value of this parameter is the path to where the software is installed.
- **DIR\_SQLANY\_MONITOR** The value of this parameter is the path to where the SQL Anywhere Monitor database (*samonitor.db*) is installed.
- **REGKEY** The value of this parameter must be a valid software installation key.
- **REGKEY\_ADD\_1** The value of this parameter must be a valid software installation key for an add-on feature such as ECC or FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.
- **REGKEY\_ADD\_2** The value of this parameter must be a valid software installation key for an add-on feature such as ECC or FIPS encryption. This property applies only if you have optional add-on features to be installed during this run of the installer.

The following example shows how to specify the SQL Anywhere Monitor install properties:

```
d:\software\monitor\setup.exe /S "/v: /qn
REGKEY=<SQLAnywhere Monitor registration key>
REGKEY_ADD_1=<an add-on software registration key>
REGKEY_ADD_2=<another add-on software registration key>
INSTALLDIR=c:\sa12"
```

In addition to setting the property values described above, the following SQL Anywhere components or features may be selected from the command-line:

Feature	Property	x86 Default	x64 Default
Administration Tools (32-bit)	AT32	1	0
Administration Tools (64-bit)	AT64	0	1
CAC Authentication	CAC *		
ECC Encryption	ECC *		
FIPS-certified Encryption	FIPS *		
High Availability	HA *		
In-Memory Mode	IM *		
MobiLink (32-bit)	ML32	1	0
MobiLink (64-bit)	ML64	0	1
QAnywhere	QA32	1	1
Read-only scale-out	SON *		

Feature	Property	x86 Default	x64 Default
Relay Server (32-bit)	RS32	1	0
Relay Server (64-bit)	RS64	0	1
Samples	SAMPLES	1	1
SQL Anywhere (32-bit)	SA32	1	0
SQL Anywhere (64-bit)	SA64	0	1
SQL Anywhere for Windows Mobile	MOBILE	1	1
SQL Anywhere Monitor (32-bit)	SM32	1	0
SQL Anywhere Monitor (64-bit)	SM64	0	1
SQL Remote (32-bit)	SR32	1	0
SQL Remote (64-bit)	SR64	0	1
UltraLite	UL	1	1

\* These features require an additional appropriate software registration key.

Set the property value to 1 to select the feature or to 0 to omit the feature. For example, to omit 32-bit MobiLink, set ML32=0. To select the 32-bit Administration Tools, set AT32=1. These properties are used to override the default selection.

You don't need to specify the selection state of a feature if the default is applicable. For example, to override the default selection on a 64-bit computer by installing the 32-bit MobiLink feature and not installing the Samples, use the following command line:

```
setup.exe "/v ml32=1 samples=0"
```

Note that property names are case insensitive.

Note that your registration key may limit the available features. The command-line switches cannot be used to override those limitations. For example, if your registration key does not allow the install of ECC Encryption, then using ECC=1 on the command-line will not select the feature.

To generate an MSI log add the following to the command line after the /v:

```
/l*v! logfile
```

In this example, `logfile` is the full path and file name of the log file. The path must already exist. Note that this switch will generate an extremely verbose log and will significantly lengthen the time required to execute the install. For information about reducing the output to the log file, see <http://msdn.microsoft.com/en-us/library/aa367988.aspx>.



In addition to a silent install, it is also possible to perform a silent uninstall. The following is an example of a command line that would do this.

```
msiexec.exe /qn /uninstall {1DFA77E6-91B2-4DCC-B8BE-98EA70705D39}
```

In the above example, you call the Microsoft Windows Installer tool directly.

- **/qn** This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.
- **/uninstall <product-code>** This parameter tells the Microsoft Windows Installer to uninstall the product associated with the specified product code. The code shown above is for the SQL Anywhere software.

The product codes for SQL Anywhere are:

- **{1DFA77E6-91B2-4DCC-B8BE-98EA70705D39}** SQL Anywhere software
- **{A8429447-7813-4717-9803-EB505ECAE698}** SQL Anywhere Client-only software
- **{2E34270B-A38A-4C97-986F-0FC4BBEBA181}** SQL Anywhere Monitor

## Requirements for deploying client applications

To deploy a client application that runs against a network database server, you must provide each end user with the following items:

- **Client application** The application software itself is independent of the database software, and so is not described here.
- **Database interface files** The client application requires the files for the database interface it uses (.NET, ADO, OLE DB, ODBC, JDBC, embedded SQL, or Open Client).
- **Connection information** Each client application needs database connection information.

The interface files and connection information required varies with the interface your application is using. Each interface is described separately in the following sections.

The simplest way to deploy clients is to use the **Deployment Wizard**. For more information, see [“The Deployment Wizard” on page 919](#).

## .NET client deployment

The simplest way to deploy .NET assemblies is to use the **Deployment Wizard**. For more information, see [“The Deployment Wizard” on page 919](#).

To complete the installation of the .NET assemblies on the client computer, you must do the following.

- Ensure Visual Studio is not running.
- Use the SetupVSPackage tool to install the .NET assemblies. SetupVSPackage requires Administrator privilege for Windows Vista and later systems. If you are running a Command Prompt, ensure it has Administrator privilege.
- For .NET 2.0/3.x, run `%SQLANY12%\Assembly\v2\SetupVSPackage.exe /install`.
- For .NET 4.0, run `%SQLANY12%\Assembly\v4\SetupVSPackage.exe /install`.
- By default, SetupVSPackage will use SQL Anywhere registry settings to locate the .NET assemblies. You can specify the location of the SQL Anywhere installation by using the `/salocation` option.

```
%SQLANY12%\Assembly\v2\SetupVSPackage.exe /install /salocation %SQLANY12%
```

The short form for `/salocation` is `/sal`.

The SetupVSPackage application updates the Global Assembly Cache and the Windows Microsoft.NET *machine.config* file. If SQL Server 2008 or later is installed on the system, SetupVSPackage also installs two mapping files called *MSSqlToSA.xml* and *SAToMSSql10.xml* to the SQL Server *DTS\MappingFiles* folder.

If you want to create your own installation, this section describes the files to deploy to the end users. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace **[LL]** with the language code (for example, **en**, **de**, **jp**, and so on).

Each .NET client computer must have the following:

- **A working .NET 2.0, 3.0, 3.5 or 4.0 installation** Microsoft .NET assemblies and instructions for their redistribution are available from Microsoft Corporation. They are not described in detail here.
- **SQL Anywhere provider for .NET Framework 2.0/3.0** The SQL Anywhere installation places the Windows assemblies for the .NET Framework versions 2.0 and 3.0 in the *Assembly\v2* subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, *bin32* or *bin64*). The following files are required.

```
iAnywhere.Data.SQLAnywhere.dll  
policy.12.0.iAnywhere.Data.SQLAnywhere.dll  
dblg[LL]12.dll  
dbicu12.dll  
dbicudt12.dll  
dbcon12.dll
```

- **SQL Anywhere provider for .NET Framework 3.5** The SQL Anywhere installation places the Windows assemblies for the .NET Framework version 3.5 in the *Assembly\v3.5* subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, *bin32* or *bin64*). The following files are required.

```
iAnywhere.Data.SQLAnywhere.v3.5.dll  
policy.12.0.iAnywhere.Data.SQLAnywhere.v3.5.dll
```

```

dblg[LL]12.dll
dbicu12.dll
dbicudt12.dll
dbcon12.dll

```

- **SQL Anywhere provider for .NET Framework 4.0** The SQL Anywhere installation places the Windows assemblies for the .NET Framework version 4.0 in the *Assembly\V4* subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system binaries directory of your SQL Anywhere installation directory (for example, *bin32* or *bin64*). The following files are required.

```

iAnywhere.Data.SQLAnywhere.v4.0.dll
policy.12.0.iAnywhere.Data.SQLAnywhere.v4.0.dll
dblg[LL]12.dll
dbicu12.dll
dbicudt12.dll
dbcon12.dll
SSDLToSA12.tt

```

*SSDLToSA12.tt* is used for generating database schema DDL for Entity Data Models. The SQL Anywhere installer copies this file to the Visual Studio 2010 directory. The user should set the DDL Generation property to this file when generating database schema DDL for Entity Data Models.

- **SQL Anywhere provider for .NET Compact Framework 2.0** The SQL Anywhere installation places the Windows Mobile assemblies for the .NET Compact Framework in *CE\Assembly\V2*. The other file is placed in the Windows Mobile binaries subdirectory of your SQL Anywhere installation directory (for example, *CE\Arm.50*). The following files are required.

```

iAnywhere.Data.SQLAnywhere.dll
iAnywhere.Data.SQLAnywhere.gac
dblg[LL]12.dll
dbicu12.dll
dbicudt12.dll

```

For more information about deploying the SQL Anywhere .NET provider, see [“SQL Anywhere .NET Data Provider deployment”](#) on page 69.

## Uninstalling the SQL Anywhere .NET provider

To uninstall the current version of the SQL Anywhere .NET provider, you must do the following.

- Ensure Visual Studio is not running.
- Use the SetupVSPackage tool to uninstall the .NET assemblies. SetupVSPackage requires Administrator privilege for Windows Vista and later systems. If you are running a Command Prompt, ensure it has Administrator privilege.
- For .NET 2.0/3.x, run `%SQLANY12%\Assembly\v2\SetupVSPackage.exe /uninstall`.
- For .NET 4.0, run `%SQLANY12%\Assembly\v4\SetupVSPackage.exe /uninstall`.

The short form for **/uninstall** is **/u**.

To uninstall all builds of the 12.0.1 version of the SQL Anywhere .NET provider, you must do the following.

- Ensure Visual Studio is not running.
- Use the SetupVSPackage tool to uninstall all builds of the .NET assemblies. SetupVSPackage requires Administrator privilege for Windows Vista and later systems. If you are running a Command Prompt, ensure it has Administrator privilege.
- For .NET 2.0/3.x, run `%SQLANY12%\Assembly\v2\SetupVSPackage.exe /uninstallall`.
- For .NET 4.0, run `%SQLANY12%\Assembly\v4\SetupVSPackage.exe /uninstallall`.

The short form for `/uninstallall` is `/ua`.

## OLE DB and ADO client deployment

The simplest way to deploy OLE DB client libraries is to use the **Deployment Wizard**.

If you want to create your own installation, this section describes the files to deploy to the end users.

Each OLE DB client computer must have the following:

- **A working OLE DB installation** OLE DB files and instructions for their redistribution are available from Microsoft Corporation. They are not described in detail here.
- **The SQL Anywhere OLE DB provider** The following table shows the files needed for a working SQL Anywhere OLE DB provider. These files should be placed in a single directory. The SQL Anywhere installation places them all in the operating-system subdirectory of your SQL Anywhere installation directory (for example, *bin32* or *bin64*). For Windows, there are two provider DLLs. The second DLL (*dboledba12*) is an assist DLL used to provide schema support.

Description	Windows
OLE DB driver file	<i>dboledb12.dll</i>
OLE DB driver file	<i>dboledba12.dll</i>
Language-resource library	<i>dblg[LL]12.dll</i>
Connect window	<i>dbcon12.dll</i>
Elevated operations agent	<i>dbelevate12.exe</i> (Windows Vista or later only)

The table above shows a file with the designation **[LL]**. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace **[LL]** with the language code (for example, **en**, **de**, **jp**, and so on).

OLE DB providers require many registry entries. You can make these by self-registering the *dboledb12.dll* and *dboledba12.dll* DLLs using the regsvr32 utility.

Note that for Windows Vista or later versions of Windows, you must include the SQL Anywhere elevated operations agent which supports the privilege elevation required when DLLs are registered or unregistered. This file is only required as part of the OLE DB provider install or uninstall procedure.

For Windows clients, it is recommended that you use Microsoft MDAC 2.7 or later.

## Customizing the OLE DB provider

When installing the OLE DB provider, the Windows Registry must be modified. Typically, this is done using the self-registration capability built into the OLE DB provider. For example, you would use the Windows regsvr32 tool to do this. A standard set of registry entries are created by the provider.

In a typical connection string, one of the components is the Provider attribute. To indicate that the SQL Anywhere OLE DB provider is to be used, you specify the name of the provider. Here is a Visual Basic example:

```
connectString = "Provider=SAOLEDB;DSN=SQL Anywhere 12 Demo"
```

With ADO and/or OLE DB, there are many other ways to reference the provider by name. Here is a C++ example in which you specify not only the provider name but also the version to use.

```
hr = db.Open(_T("SAOLEDB.12"), &dbinit);
```

The provider name is looked up in the registry. If you were to examine the registry on your computer system, you would find an entry in HKEY\_CLASSES\_ROOT for SAOLEDB.

```
[HKEY_CLASSES_ROOT\SAOLEDB]
@"SQL Anywhere OLE DB Provider"
```

It has two subkeys that contain a class identifier (Clsid) and current version (CurVer) for the provider. Here is an example.

```
[HKEY_CLASSES_ROOT\SAOLEDB\Clsid]
@="{41dfe9f7-db91-11d2-8c43-006008d26a6f}"
```

```
[HKEY_CLASSES_ROOT\SAOLEDB\CurVer]
@"SAOLEDB.12"
```

There are several more similar entries. They are used to identify a specific instance of an OLE DB provider. If you look up the Clsid in the registry under HKEY\_CLASSES\_ROOT\CLSID and examine the subkeys, you see that one of the entries identifies the location of the provider DLL.

```
[HKEY_CLASSES_ROOT\CLSID\
{41dfe9f3-db91-11d2-8c43-006008d26a6f}\
InprocServer32]
```

```
@="c:\sa12\bin64\dboledb12.dll"
"ThreadingModel"="Both"
```

The problem here is that the structure is very monolithic. If you were to uninstall the SQL Anywhere software from your system, the OLE DB provider registry entries would be removed from your registry

and then the provider DLL would be removed from your hard drive. Any applications that depend on the provider would no longer work.

Similarly, if applications from different vendors all use the same OLE DB provider, then each installation of the same provider would overwrite the common registry settings. The version of the provider that you intended your application to work with would be supplanted by another newer (or older!) version of the provider.

Clearly, the instability that could arise from this situation is undesirable. To address this problem, the SQL Anywhere OLE DB provider can be customized. Here are the steps involved in creating a custom version of the OLE DB provider.

### Customize the OLE DB provider

In this exercise, you generate a unique set of GUIDs, choose a unique provider name, and choose unique DLL names. These three things will help you create a unique OLE DB provider which you can deploy with your application.

1. Make a copy of the sample registration file shown below. It is listed after these steps because it is quite lengthy. The file name should have a *.reg* suffix. The names of the registry values are case sensitive.
2. Use the Microsoft Visual Studio `uuidgen` utility to create 4 sequential UUIDs (GUIDs).

```
uuidgen -n4 -s -x >oledbguids.txt
```

3. The 4 UUIDs or GUIDs are assigned in the following sequence:
  - a. The Provider class ID (GUID1 below).
  - b. The Enum class ID (GUID2 below).
  - c. The ErrorLookup class ID (GUID3 below).
  - d. The Provider Assist class ID (GUID4 below). This last GUID is not used in Windows Mobile deployments.

It is important that they be sequential (that is what `-x` in the `uuidgen` command line does for you). Each GUID should appear similar to the following.

Name	GUID
GUID1	41dfe9f3-db92-11d2-8c43-006008d26a6f
GUID2	41dfe9f4-db92-11d2-8c43-006008d26a6f
GUID3	41dfe9f5-db92-11d2-8c43-006008d26a6f
GUID4	41dfe9f6-db92-11d2-8c43-006008d26a6f

Note that it is the first part of the GUID (for example, 41dfe9f3) that is incrementing.

- Use the search/replace capability of an editor to change all the GUID1, GUID2, GUID3, and GUID4 in the text to the corresponding GUID (for example, GUID1 would be replaced by 41dfe9f3-db92-11d2-8c43-006008d26a6f if that was the GUID generated for you by uuidgen).
- Decide on your Provider name. This is the name that you will use in your application in connection strings, and so on (for example, Provider=SQLAny). Do not use any of the following names. These names are used by SQL Anywhere.

Version 10 or later	Version 9 or earlier
SAOLEDB	ASAProv
SAErrorLookup	ASAEErrorLookup
SAEnum	ASAEEnum
SAOLEDBA	ASAProvA

- Use the search/replace capability of an editor to change all the occurrences of the string SQLAny to the provider name that you have chosen. This includes all those places where SQLAny may be a substring of a longer string (for example, SQLAnyEnum).

Suppose you chose Acme for your provider name. The names that will appear in the HKEY\_CLASSES\_ROOT registry hive are shown in the following table along with the SQL Anywhere names (for comparison).

SQL Anywhere provider	Your custom provider
SAOLEDB	Acme
SAErrorLookup	AcmeErrorLookup
SAEnum	AcmeEnum
SAOLEDBA	AcmeA

- Make copies of the SQL Anywhere provider DLLs (*dboledb12.dll* and *dboledba12.dll*) under different names. Note that there is no *dboledba12.dll* for Windows Mobile.

```
copy dboledb12.dll myoledb12.dll
copy dboledba12.dll myoledba12.dll
```

A special registry key will be created by the script that is based on the DLL name that you choose. It is important that the name be different from the standard DLL names (such as *dboledb12.dll* or *dboledba12.dll*). If you name the provider DLL *myoledb12* then the provider will look up a registry entry in HKEY\_CLASSES\_ROOT with that same name. The same is true of the provider schema assist DLL. If you name the DLL *myoledba12* then the provider will look up a registry entry in HKEY\_CLASSES\_ROOT with that same name. It is important that the name you choose is unique and is unlikely to be chosen by anyone else. Here are some examples.

DLL name(s) chosen	Corresponding HKEY_CLASSES_ROOTname
<i>myoledb12.dll</i>	HKEY_CLASSES_ROOT\myoledb12
<i>myoledba12.dll</i>	HKEY_CLASSES_ROOT\myoledba12
<i>acmeOledb.dll</i>	HKEY_CLASSES_ROOT\acmeOledb
<i>acmeOledba.dll</i>	HKEY_CLASSES_ROOT\acmeOledba
<i>SAcustom.dll</i>	HKEY_CLASSES_ROOT\SAcustom
<i>SAcustomA.dll</i>	HKEY_CLASSES_ROOT\SAcustomA

8. Use the search/replace capability of an editor to change all the occurrences of *myoledb12* and *myoledba12* in the registry script to the two DLL names you have chosen.
9. Use the search/replace capability of an editor to change all the occurrences of *d:\myopath\bin32\* in the registry script to the installed location for the DLLs. Be sure to use a pair of slashes to represent a single slash. This step must be customized at the time of your application install.
10. Save the registry script to disk and run it.
11. Give your new provider a try. Do not forget to change your ADO / OLE DB application to use the new provider name.

Here is the listing of the registry script that is to be modified.

```

REGEDIT4
; Special registry entries for a private OLE DB provider.
[HKEY_CLASSES_ROOT\myoledb12]
@="Custom SQL Anywhere OLE DB Provider 12.0"
[HKEY_CLASSES_ROOT\myoledb12\Clsid] @="{GUID1}"
; Data of the following GUID must be 3 greater than the
; previous, for example, 41dfe9f3 + 3 => 41dfe9ee.
[HKEY_CLASSES_ROOT\myoledba12]
@="Custom SQL Anywhere OLE DB Provider 12.0"
[HKEY_CLASSES_ROOT\myoledba12\Clsid] @="{GUID4}"
; Current version (or version independent prog ID)
; entries (what you get when you have "SQLAny"
; instead of "SQLAny.12")
[HKEY_CLASSES_ROOT\SQLAny]
@="SQL Anywhere OLE DB Provider"
[HKEY_CLASSES_ROOT\SQLAny\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAny\CurVer]
@="SQLAny.12"
[HKEY_CLASSES_ROOT\SQLAnyEnum]
@="SQL Anywhere OLE DB Provider Enumerator"
[HKEY_CLASSES_ROOT\SQLAnyEnum\Clsid]
@="{GUID2}" [HKEY_CLASSES_ROOT\SQLAnyEnum\CurVer]
@="SQLAnyEnum.12" [HKEY_CLASSES_ROOT\SQLAnyErrorLookup]
@="SQL Anywhere OLE DB Provider Extended Error Support"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\CurVer]

```



```

@="SQLAnyErrorLookup.12"
[HKEY_CLASSES_ROOT\SQLAnyA]
@="SQL Anywhere OLE DB Provider Assist"
[HKEY_CLASSES_ROOT\SQLAnyA\Clsid]
@="{GUID4}"
[HKEY_CLASSES_ROOT\SQLAnyA\CurVer]
@="SQLAnyA.12"
; Standard entries (Provider=SQLAny.12)
[HKEY_CLASSES_ROOT\SQLAny.12]
@="Sybase SQL Anywhere OLE DB Provider 12.0"
[HKEY_CLASSES_ROOT\SQLAny.12\Clsid]
@="{GUID1}"
[HKEY_CLASSES_ROOT\SQLAnyEnum.12]
@="Sybase SQL Anywhere OLE DB Provider Enumerator 12.0"
[HKEY_CLASSES_ROOT\SQLAnyEnum.12\Clsid]
@="{GUID2}"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.12]
@="Sybase SQL Anywhere OLE DB Provider Extended Error Support 12.0"
[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.12\Clsid]
@="{GUID3}"
[HKEY_CLASSES_ROOT\SQLAnyA.12]
@="Sybase SQL Anywhere OLE DB Provider Assist 12.0"
[HKEY_CLASSES_ROOT\SQLAnyA.12\Clsid]
@="{GUID4}"
; SQLAny (Provider=SQLAny.12)
[HKEY_CLASSES_ROOT\CLSID\{GUID1}]
@="SQLAny.12"
"OLEDB_SERVICES"=dword:ffffffff
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors]
@="Extended Error Service"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors\{GUID3}]
@="Sybase SQL Anywhere OLE DB Provider Error Lookup"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\InprocServer32]
@="d:\myopath\bin32\myoledb12.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\OLE DB Provider]
@="Sybase SQL Anywhere OLE DB Provider 12.0"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ProgID]
@="SQLAny.12"
[HKEY_CLASSES_ROOT\CLSID\{GUID1}\VersionIndependentProgID]
@="SQLAny"
; SQLAnyErrorLookup
[HKEY_CLASSES_ROOT\CLSID\{GUID3}]
@="Sybase SQL Anywhere OLE DB Provider Error Lookup 12.0"
@="SQLAnyErrorLookup.12"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\InprocServer32]
@="d:\myopath\bin32\myoledb12.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\ProgID]
@="SQLAnyErrorLookup.12"
[HKEY_CLASSES_ROOT\CLSID\{GUID3}\VersionIndependentProgID]
@="SQLAnyErrorLookup"
; SQLAnyEnum [HKEY_CLASSES_ROOT\CLSID\{GUID2}]
@="SQLAnyEnum.12"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\InprocServer32]
@="d:\myopath\bin32\myoledb12.dll"
"ThreadingModel"="Both"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\OLE DB Enumerator]
@="Sybase SQL Anywhere OLE DB Provider Enumerator"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\ProgID]
@="SQLAnyEnum.12"
[HKEY_CLASSES_ROOT\CLSID\{GUID2}\VersionIndependentProgID]
@="SQLAnyEnum"
; SQLAnyA [HKEY_CLASSES_ROOT\CLSID\{GUID4}]

```

```
@="SQLAnyA.12"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\InprocServer32]  
@="d:\mypath\bin32\myoledba12.dll"  
"ThreadingModel"="Both"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\ProgID]  
@="SQLAnyA.12"  
[HKEY_CLASSES_ROOT\CLSID\{GUID4}\VersionIndependentProgID]  
@="SQLAnyA"
```

## ODBC client deployment

The simplest way to deploy ODBC clients is to use the **Deployment Wizard**.

Each ODBC client computer must have the following:

- **ODBC Driver Manager** Microsoft provides an ODBC Driver Manager for Windows operating systems. SQL Anywhere includes an ODBC Driver Manager for Linux, Unix, and Mac OS X. There is no ODBC Driver Manager for Windows Mobile. ODBC applications can run without a driver manager but, on platforms for which an ODBC driver manager is available, this is not recommended.
- **Connection information** The client application must have access to the information needed to connect to the server. This information is typically included in an ODBC data source.
- **ODBC driver** The SQL Anywhere ODBC driver must be installed.

### See also

- [“The Deployment Wizard” on page 919](#)
- [“ODBC driver required files” on page 936](#)

## ODBC driver required files

The following table shows the files needed for a working SQL Anywhere ODBC driver. These files should be placed in a single directory. The SQL Anywhere installation places them all in the operating-system subdirectory of your SQL Anywhere installation directory (for example, *bin32* or *bin64*).

The multithreaded version of the ODBC driver for Linux, Unix, and Mac OS X platforms is indicated by "MT".

<b>Platform</b>	<b>Required files</b>
Windows	<i>dbodbc12.dll</i> <i>dbcon12.dll</i> <i>sacshelp12.chm</i> (optional) <i>dbicu12.dll</i> <i>dbicudt12.dll</i> <i>dblg[LL]12.dll</i> <i>dbelevate12.exe</i>
Windows Mobile	<i>dbodbc12.dll</i> <i>dbicu12.dll</i> (optional) <i>dbicudt12.dat</i> (optional) <i>dblg[LL]12.dll</i>
Linux, Solaris, HP-UX	<i>libdbodbc12.so.1</i> <i>libdbodbc12_n.so.1</i> <i>libdbodm12.so.1</i> <i>libdbtasks12.so.1</i> <i>libdbicu12.so.1</i> <i>libdbicudt12.so.1</i> <i>dblg[LL]12.res</i>

Platform	Required files
Linux, Solaris, HP-UX MT	<i>libdbodbc12.so.1</i> <i>libdbodbc12_r.so.1</i> <i>libdbodm12.so.1</i> <i>libdbtasks12_r.so.1</i> <i>libdbicu12_r.so.1</i> <i>libdbicudt12.so.1</i> <i>dblg[LL]12.res</i>
AIX	<i>libdbodbc12.so</i> <i>libdbodbc12_n.so</i> <i>libdbodm12.so</i> <i>libdbtasks12.so</i> <i>libdbicu12.so</i> <i>libdbicudt12.so</i> <i>dblg[LL]12.res</i>
AIX MT	<i>libdbodbc12.so</i> <i>libdbodbc12_r.so</i> <i>libdbodm12.so</i> <i>libdbtasks12_r.so</i> <i>libdbicu12_r.so</i> <i>libdbicudt12.so</i> <i>dblg[LL]12.res</i>

Platform	Required files
Mac OS X	<i>dbodbc12.bundle</i> <i>libdbodbc12.dylib</i> <i>libdbodbc12_n.dylib</i> <i>libdbodm12.dylib</i> <i>libdbtasks12.dylib</i> <i>libdbicu12.dylib</i> <i>libdbicudt12.dylib</i> <i>dblg[LL]12.res</i>
Mac OS X MT	<i>dbodbc12_r.bundle</i> <i>libdbodbc12.dylib</i> <i>libdbodbc12_r.dylib</i> <i>libdbodm12.dylib</i> <i>libdbtasks12_r.dylib</i> <i>libdbicu12_r.dylib</i> <i>libdbicudt12.dylib</i> <i>dblg[LL]12.res</i>

### Notes

- For Linux and Solaris platforms, you should create a link to the *.so.1* files. The link name should match the file name with the ".1" version suffix removed.
- There are multithreaded (MT) versions of the ODBC driver for Linux, Unix, and Mac OS X platforms. The file names contain the "\_r" suffix. Deploy these files if your application requires them.
- For Windows, a driver manager is included with the operating system. For Linux, Unix, and Mac OS X, SQL Anywhere provides a driver manager. The file name begins with *libdbodm12*.
- For Windows, the *sacshelp12.chm* help file can be installed to provide local context sensitive help when using the Microsoft ODBC Data Source Administrator. If the help file is not locatable, then an attempt will be made to go to DocCommentXchange using the system's default Internet browser.
- Note that for Windows Vista or later versions of Windows, you must include the SQL Anywhere elevated operations agent (*dbelevate12.exe*) which supports the privilege elevation required to register

or unregister the ODBC driver. This file is only required as part of the ODBC driver install or uninstall procedure.

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **jp**, and so on).
- For Windows, the **ODBC Configuration for SQL Anywhere** and **Connect to SQL Anywhere** window support code (*dbcon12.dll*) is needed if your end users will create their own data sources, if they need to enter user IDs and passwords when connecting to the database, or if they need to display the Connect window for any other purpose.

## ODBC driver configuration

In addition to copying the ODBC driver files onto disk, your installation program must also make a set of registry entries to install the ODBC driver properly.

### Windows

The SQL Anywhere installer makes changes to the Windows Registry to identify and configure the ODBC driver. If you are building an installation program for your end users, you should make the same registry settings.

The simplest way to do this is to use the self-registering capability of the ODBC driver. You use the `regsvr32` utility on Windows or the `regsvrce` utility on Windows Mobile. Note that for 64-bit versions of Windows, you can register both the 64-bit and 32-bit versions of the ODBC driver. By using the self-registering feature of the ODBC driver, you are ensured that the proper registry entries are created.

You should give custom names to your 32-bit and 64-bit versions of the SQL Anywhere ODBC driver. This facilitates the installation and registration of multiple independent copies of the SQL Anywhere ODBC driver using `regsvr32`, and prevents your registry settings from being overwritten if another application install registers the SQL Anywhere ODBC driver.

To customize the names of your 32-bit and 64-bit versions of the ODBC driver, open a command prompt and rename them as follows where *custom-name* is a meaningful string such as your company name:

```
ren "%SQLANY12%\bin32\dbodbc12.dll" dbodbc12custom-name.dll
ren "%SQLANY12%\bin64\dbodbc12.dll" dbodbc12custom-name.dll
```

Make sure to preserve the `dbodbc12` prefix when renaming the files.

To register the 32-bit and 64-bit versions of the ODBC driver using the custom names, issue the following commands:

```
regsvr32 "%SQLANY12%\bin32\dbodbc12custom-name.dll"
regsvr32 "%SQLANY12%\bin64\dbodbc12custom-name.dll"
```

You can use the `regedit` utility to inspect the registry entries created by the ODBC driver.

The SQL Anywhere ODBC driver is identified to the system by a set of registry values in the following registry key:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    ODBC\
      ODBCINST.INI\
        SQL Anywhere 12
```

Sample values for 32-bit Windows are shown below:

Value name	Value type	Value data
Driver	String	<i>C:\Program Files\SQL Anywhere 12\bin32\dbodbc12custom-name.dll</i>
Setup	String	<i>C:\Program Files\SQL Anywhere 12\bin32\dbodbc12custom-name.dll</i>

There is also a registry value in the following key:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    ODBC\
      ODBCINST.INI\
        ODBC Drivers
```

The value is as follows:

Value name	Value type	Value data
SQL Anywhere 12 - <i>custom-name</i>	String	Installed

## 64-bit Windows

For 64-bit Windows, the 32-bit ODBC driver registry entries ("SQL Anywhere 12 - *custom-name*" and "ODBC Drivers") are located under the following key:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    Wow6432Node\
      ODBC\
        ODBCINST.INI
```

To view these entries, you must be using a 64-bit version of regedit. If you cannot locate Wow6432Node on 64-bit Windows, then you are using the 32-bit version of regedit.

## Third party ODBC drivers

If you are using a third-party ODBC driver on an operating system other than Windows, consult the documentation for that driver on how to configure the ODBC driver.

## Connection information deployment

ODBC client connection information is generally deployed as an ODBC data source. You can deploy an ODBC data source in one of the following ways:

- **Programmatically** Add a data source description to your end-user's registry or ODBC initialization files.
- **Manually** Provide your end users with instructions, so that they can create an appropriate data source on their own computer.

On Windows, you create a data source manually using the ODBC Data Source Administrator, from the User DSN tab or the System DSN tab. The SQL Anywhere ODBC driver displays the configuration window for entering settings. Data source settings include the location of the database file, the name of the database server, and any start up parameters and other options. Context sensitive help is accessed by clicking the **Help** button. Help text is provided locally using *sacshelp12.chm*. If this file is not present, then help is provided over the Internet from DocCommentXchange using the system's default Internet browser.

On Unix platforms, you can create a data source manually using the SQL Anywhere dbdsn utility. Data source settings include the location of the database file, the name of the database server, and any start up parameters and other options.

This section provides you with the information you need to know for either approach.

### Types of data source (Windows)

There are three kinds of data sources: User Data Sources, System Data Sources, and File Data Sources.

1. User Data Source definitions are stored in the part of the registry containing settings for the specific user currently logged on to the system.
2. System Data Sources, however, are available to all users and to Windows services, which run regardless of whether a user is logged onto the system or not. Given a correctly configured System Data Source named MyApp, any user can use that ODBC data source by providing DSN=MyApp in the ODBC connection string.
3. File Data Sources are not held in the registry, but are stored on disk. A connection string must provide a FileDSN connection parameter to use a File Data Source. The default location of File Data Sources is specified by the **HKEY\_CURRENT\_USER\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir** registry entry or by the **HKEY\_LOCAL\_MACHINE\Software\ODBC\odbc.ini\ODBC File DSN\DefaultDSNDir** registry entry (if the former is not defined). The path can be included in the FileDSN connection parameter to help locate the File Data Source when it is located elsewhere.

### Data source registry entries (Windows)

User and System Data Source definitions are stored in the Windows registry. The simplest way to ensure the correct creation of registry entries for data source definitions is to use the SQL Anywhere dbdsn utility to create them.



Otherwise, you must create a set of registry values in a particular registry key.

For User Data Sources, the registry key is as follows:

```
HKEY_CURRENT_USER\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        user-data-source-name
```

For System Data Sources, the registry key is as follows:

```
HKEY_LOCAL_MACHINE\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        system-data-source-name
```

The key contains a set of registry values, each of which corresponds to a connection parameter (except for the Driver string). The Driver string is automatically added to the registry by the Microsoft ODBC Driver Manager when a Data Source is created using the Microsoft ODBC Data Source Administrator or the dbdsn utility. For example, the SQL Anywhere 12 Demo key corresponding to the SQL Anywhere 12 Demo System Data Source Name (DSN) contains the following settings for 32-bit Windows:

Value name	Value type	Value data
AutoStop	String	YES
DatabaseFile	String	<i>C:\Users\Public\Documents\SQL Anywhere 12\Samples\demo.db</i>
Description	String	SQL Anywhere 12 Sample Database
Driver	String	<i>C:\Program Files\SQL Anywhere 12\bin32\dbodbc12.dll</i>
Password	String	sql
ServerName	String	demo12
StartLine	String	<i>C:\Program Files\SQL Anywhere 12\bin32\dbeng12.exe</i>
UserID	String	DBA

**Note**

It is recommended that you include the ServerName parameter in connection strings for deployed applications. This ensures that the application connects to the correct server if a computer is running multiple SQL Anywhere database servers and can help prevent timing-dependent connection failures.

In these entries, for 64-bit Windows, *bin32* would be replaced by *bin64*.

On 64-bit Windows, there is only one registry entry for User Data Sources that is shared by both 64-bit and 32-bit applications. For these types of data sources, it is recommended that you remove the path in the

Driver string. If the path designates the 64-bit ODBC driver, then a 32-bit application that connects/disconnects often will incur a performance penalty (as well as a memory leak) when using the Microsoft ODBC Driver Manager. Similarly, If the path designates the 32-bit ODBC driver, then a 64-bit application that connects/disconnects often will incur a similar penalty. At the time of writing, this is a flaw in the Microsoft ODBC Driver Manager. For this reason, System Data Sources are a better choice for 64-bit Windows systems.

In addition, you must add the data source name to the list of data sources in the registry. For User Data Sources, you use the following key:

```
HKEY_CURRENT_USER\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        ODBC Data Sources
```

For System Data Sources, use the following key:

```
HKEY_LOCAL_MACHINE\  
  SOFTWARE\  
    ODBC\  
      ODBC.INI\  
        ODBC Data Sources
```

The value associates each data source with an ODBC driver. The value name is the data source name, and the value data is the ODBC driver name. For example, the System Data Source installed by SQL Anywhere is named SQL Anywhere 12 Demo, and has the following value:

Value name	Value type	Value data
SQL Anywhere 12 Demo	String	SQL Anywhere 12

**Caution**

ODBC Data Source Name settings are easily viewed. User Data Source configurations can contain sensitive database settings such as a user's ID and password. These settings are stored in the registry in plain text, and can be viewed using the Windows Registry editors *regedit.exe* or *regedt32.exe*, which are provided by Microsoft with the operating system. You can choose to encrypt passwords, or require users to enter them when connecting.

**Required and optional connection parameters**

You can identify the data source name in an ODBC connection string in this manner,

```
DSN=UserDataSourceName
```

On Windows, when a DSN parameter is provided in the connection string, the User Data Source definitions in the Windows Registry are searched, followed by System Data Sources. File Data Sources are searched only when FileDSN is provided in the ODBC connection string.

The following table illustrates the implications to the user and the application developer when a data source exists and is included in the application's connection string as a DSN or FileDSN parameter.

When the data source...	The connection string must also identify...	The user must supply...
Contains the ODBC driver name and location; the name of the database file/server; startup parameters; and the user ID and password.	No additional information	No additional information.
Contains the ODBC driver name and location; the name of the database file/server; startup parameters.	No additional information	User ID and password if not provided in the ODBC data source.
Contains only the name and location of the ODBC driver.	The name of the database file (DBF=) and/or the database server (SERVER=). Optionally, it may contain other connection parameters such as Userid (UID=) and PASSWORD (PWD=).	User ID and password if not provided in the ODBC data source or ODBC connection string.
Does not exist	The name of the ODBC driver to be used (Driver=) and the database name (DBN=), the database file (DBF=), and/or the database server (SERVER=). Optionally, it may contain other connection parameters such as Userid (UID=) and PASSWORD (PWD=).	User ID and password if not provided in the ODBC connection string.

**See also**

- “SQL Anywhere database connections” [[SQL Anywhere Server - Database Administration](#)]
- The Open Database Connectivity (ODBC) SDK, available from Microsoft.

## Embedded SQL client deployment

The simplest way to deploy embedded SQL clients is to use the **Deployment Wizard**.

Deploying embedded SQL clients involves the following:

- **Installed files** Each client computer must have the files required for a SQL Anywhere embedded SQL client application.
- **Connection information** The client application must have access to the information needed to connect to the server. This information may be included in an ODBC data source.

**See also**

- [“The Deployment Wizard” on page 919](#)

## Required files for embedded SQL clients

The following table shows which files are needed for embedded SQL clients.

Description	Windows	Linux / Unix	Mac OS X
Interface library	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>
optional encryption support	<i>dbecc12.dll</i>	<i>libdbecc12_r.so</i>	<i>libdbecc12_r.dylib</i>
optional encryption support	<i>dbfips12.dll</i>	<i>libdbfips12_r.so</i>	<i>libdbfips12_r.dylib</i>
optional encryption support	<i>dbrsa12.dll</i>	<i>libdbrsa12_r.so</i>	<i>libdbrsa12_r.dylib</i>
Thread support library	N/A	<i>libdbtasks12_r.so</i>	<i>libdbtasks12_r.dylib</i>
Language resource library	<i>dblg[LL]12.dll</i>	<i>dblg[LL]12.res</i>	<i>dblg[LL]12.res</i>
Connect window	<i>dbcon12.dll</i>	N/A	N/A

**Notes**

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **jp**, and so on).
- For non-multithreaded applications on Linux/Unix, use *libdblib12.so* and *libdbtasks12.so*.
- For non-multithreaded applications on Mac OS X, use *libdblib12.dylib* and *libdbtasks12.dylib*.
- If the client application uses encryption then the appropriate encryption support (*dbecc12.dll*, *dbfips12.dll*, or *dbrsa12.dll*) should also be included.
- If the client application uses an ODBC data source to hold the connection parameters, your end user must have a working ODBC installation. Instructions for deploying ODBC are included in the Microsoft ODBC SDK.
- For Windows, the **ODBC Configuration for SQL Anywhere** and **Connect to SQL Anywhere** window support code (*dbcon12.dll*) is needed if your end users will create their own data sources, if

they need to enter user IDs and passwords when connecting to the database, or if they need to display the Connect window for any other purpose.

**See also**

- [“ODBC client deployment” on page 936](#)

## Connection information

You can deploy embedded SQL connection information in one of the following ways:

- **Manual** Provide your end users with instructions for creating an appropriate data source on their computer.
- **File** Distribute a file that contains connection information in a format that your application can read.
- **ODBC data source** You can use an ODBC data source to hold connection information.

## JDBC client deployment

You must install a Java Runtime Environment (JRE) to use JDBC. Version 1.6.0 or later is recommended.

In addition to a JRE, each JDBC client requires a SQL Anywhere JDBC driver or jConnect.

### SQL Anywhere JDBC 4.0 driver

To deploy the SQL Anywhere JDBC 4.0 driver, you must deploy the following files:

- *sajdbc4.jar* This must be in the application's classpath. This file is located in the SQL Anywhere installation *java* folder.
- The JDBC driver support files.

### SQL Anywhere JDBC 3.0 driver

To deploy the SQL Anywhere JDBC 3.0 driver, you must deploy the following files:

- *sajdbc.jar* This must be in the application's classpath. This file is located in the SQL Anywhere installation *java* folder.
- The JDBC driver support files.

### jConnect JDBC driver

To deploy the jConnect JDBC driver, you must deploy the following files:

- The jConnect driver files. For a version of the jConnect software and the jConnect documentation, see <http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect>.
- When you use a TDS client (either Open Client or jConnect based), you have the option of sending the connection password in clear text or in encrypted form. The latter is done by performing a TDS

encrypted password handshake. The handshake involves using private/public key encryption. The support for generating the RSA private/public key pair and for decrypting the encrypted password is included in a special library. The library file must be locatable by the SQL Anywhere server in its system path. For Windows, this file is called *dbrsakp12.dll*. There are both 64-bit and 32-bit versions of the DLL. On Linux and Unix environments, the file is a shared library called *libdbrsakp12.so*. On Mac OS X, the file is a shared library called *libdbrsakp12.dylib*. The file is not necessary if you do not use this feature.

### JDBC database connection URL

Your Java application needs a URL to connect to the database. This URL specifies the driver, the computer to use, and the port on which the database server is listening.

### See also

- [“SQL Anywhere JDBC driver required files” on page 948](#)
- [“Supplying a URL to the driver” on page 410](#)

## SQL Anywhere JDBC driver required files

The following table shows the files needed to support the SQL Anywhere JDBC driver. These files should be placed in a single directory. The SQL Anywhere installation places them all in the operating-system subdirectory of your SQL Anywhere installation directory (for example, *bin32*, *bin64*, *lib32*, or *lib64*). The choice of 32-bit or 64-bit files depends on the bitness of your installed Java VM.

The JDBC driver support files for all platforms are multithreaded.

Platform	Required files
Windows	<i>dbjdbc12.dll</i> <i>dbicu12.dll</i> <i>dbicudt12.dll</i> <i>dblg[LL]12.dll</i>
Linux, Solaris, HP-UX	<i>libdbjdbc12.so.1</i> <i>libdbtasks12_r.so.1</i> <i>libdbicu12_r.so.1</i> <i>libdbicudt12.so.1</i> <i>dblg[LL]12.res</i>

Platform	Required files
AIX	<i>libdbjdbc12.so</i> <i>libdbtasks12_r.so</i> <i>libdbicu12_r.so</i> <i>libdbicudt12.so</i> <i>dblg[LL]12.res</i>
Mac OS X	<i>libdbjdbc12.dylib</i> <i>libdbtasks12_r.dylib</i> <i>libdbicu12_r.dylib</i> <i>libdbicudt12.dylib</i> <i>dblg[LL]12.res</i>

### Notes

- For Linux and Solaris platforms, you should create a link to the *.so.l* files. The link name should match the file name with the ".1" version suffix removed.
- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **jp**, and so on).

## PHP client deployment

To deploy the SQL Anywhere PHP extension, you must install the following components on the target platform:

- The PHP 5 binaries for your platform, which are available for download at <http://www.php.net>. For Windows platforms, the thread-safe version of PHP must be used with the SQL Anywhere PHP extension.
- A web server such as Apache HTTP Server if you want to run PHP scripts within a web server. SQL Anywhere can be run on the same computer as the web server, or on a different computer.
- SQL Anywhere provides prebuilt PHP extensions for PHP versions 5.1.1 to 5.2.11 and 5.3.0 to 5.3.5. At the time of writing, PHP version 5.2.11 and 5.3.5 were the most recent stable releases.
- Supporting SQL Anywhere shared objects or libraries.

The following table summarizes the files required for PHP clients.

Description	Windows	Linux / Unix	Mac OS X
PHP installation (third-party)	<i>php.exe</i>	<i>php</i>	<i>php</i>
PHP 5.1.x calls	<i>php-5.1.[1-6]_sqlanywhere.dll</i>	<i>php-5.1.[1-6]_sqlanywhere_r.so</i> or build from source code	Build from source code
PHP 5.2.x calls	<i>php-5.2.[0-11]_sqlanywhere.dll</i> <i>php-5.2.11_sqlanywhere_nts.dll</i>	<i>php-5.2.[0-11]_sqlanywhere_r.so</i> or build from source code	Build from source code
PHP 5.3.x calls	<i>php-5.3.[0-5]_sqlanywhere.dll</i> <i>php-5.3.[0-5]_sqlanywhere_nts.dll</i>	<i>php-5.3.[0-5]_sqlanywhere_r.so</i> or build from source code	Build from source code
SQL Anywhere C API runtime	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB (threaded)	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>
Thread support library	N/A	<i>libdbtasks12_r.so</i>	<i>libdbtasks12_r.dylib</i>
optional encryption support	<i>dbecc12.dll</i>	<i>libdbecc12_r.so</i>	<i>libdbecc12_r.dylib</i>
optional encryption support	<i>dbfips12.dll</i>	<i>libdbfips12_r.so</i>	<i>libdbfips12_r.dylib</i>
optional encryption support	<i>dbrsa12.dll</i>	<i>libdbrsa12_r.so</i>	<i>libdbrsa12_r.dylib</i>
Language resource library	<i>dblg[LL]12.dll</i>	<i>dblg[LL]12.res</i>	<i>dblg[LL]12.res</i>
Connect window	<i>dbcon12.dll</i>	N/A	N/A

### Notes

- For the latest SQL Anywhere PHP drivers, see <http://www.sybase.com/detail?id=1019698>.
- The *php-5.x.y\_sqlanywhere\_nts.dll* files are non-thread-safe version of the PHP extensions.



- A language resource library file should also be included. The table above shows files with the designation **[LL]**. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace **[LL]** with the language code (for example, **en**, **de**, **jp**, and so on).
- If the client application uses encryption then the appropriate encryption support should also be included.
- If the client application uses an ODBC data source to hold the connection parameters, your end user must have a working ODBC installation. Instructions for deploying ODBC are included in the Microsoft ODBC SDK.
- For Windows, the **ODBC Configuration for SQL Anywhere** and **Connect to SQL Anywhere** window support code (*dbcon12.dll*) is needed if your end users will create their own data sources, if they need to enter user IDs and passwords when connecting to the database, or if they need to display the Connect window for any other purpose.
- For more information about installing PHP, see <http://www.sybase.com/detail?id=1057714>.

#### See also

- [“ODBC client deployment” on page 936](#)

## PHP extensions

On Windows, SQL Anywhere includes thread-safe extensions for PHP versions 5.1.1 to 5.2.11 and 5.3.0 to 5.3.5. A thread-safe version of PHP must be used with the SQL Anywhere PHP extension. The extension file names for the supported PHP versions follow this pattern:

```
php-5.x.y_sqlanywhere.dll
```

On Windows, SQL Anywhere also includes non-thread-safe extensions for PHP versions 5.2.11 and 5.3.0 to 5.3.5. A non-thread-safe version of PHP may be used with multi-processed but not multi-threaded web servers such as Apache and IIS. The extension file names for the supported PHP versions follow this pattern:

```
php-5.x.y_sqlanywhere_nts.dll
```

On Linux and Solaris, SQL Anywhere includes both 64-bit and 32-bit versions of the extensions for PHP versions 5.1.1 to 5.2.11 and 5.3.0 to 5.3.5. It also includes both threaded and non-threaded extensions. If you are using the CGI version of PHP or if you are using Apache 1.x, use the non-threaded extension. If you are using Apache 2.x, use the threaded extension. The extension file names for the supported PHP versions follow this pattern:

```
php-5.x.y_sqlanywhere[_r].so
```

The "5.x.y" represents the PHP version (for example, 5.2.11). For Linux and Solaris, the threaded version of the PHP extension has *\_r* appended to the file name. Windows versions are implemented as Dynamic Link Libraries and Linux/Solaris versions are implemented as Shared Objects. If you are using a later

version of PHP, then you can check the Internet for the latest SQL Anywhere PHP drivers at <http://www.sybase.com/detail?id=1019698>.

## Installing the PHP extension on Windows

To use the SQL Anywhere PHP extension on Windows, you must copy the DLL from the SQL Anywhere installation directory and add it to your PHP installation. Optionally, you can add an entry to your PHP initialization file to load the extension, so you do not need to load it manually in each script.

### Install the PHP extension on Windows

1. Locate the *php.ini* file for your PHP installation, and open it in a text editor. Locate the line that specifies the location of the **extension\_dir** directory. If **extension\_dir** is not set to any specific directory, it is a good idea to set it to point to an isolated directory for better system security.
2. Copy the file *php-5.x.y\_sqlanywhere.dll* from the *Bin32* subdirectory of your SQL Anywhere installation to the directory specified by the **extension\_dir** entry in the *php.ini* file.

#### Note

The string *5.x.y* is the PHP version number corresponding to the version that you have installed.

If your version of PHP is more recent than the SQL Anywhere PHP extensions provided by SQL Anywhere, try using the most recent extension provided. Note that a version 5.2.x SQL Anywhere PHP extension will not work with a version 5.3.x PHP.

3. Add the following line to the Dynamic Extensions section of the *php.ini* file to load the SQL Anywhere PHP driver automatically.

```
extension=php-5.x.y_sqlanywhere.dll
```

where *5.x.y* reflects the version number of the SQL Anywhere PHP extension copied in the previous step.

Save and close *php.ini*.

An alternative to automatically loading the PHP driver is to load it manually in each script that requires it. See [“Configuration of the SQL Anywhere PHP extension” on page 954](#).

4. Make sure that the *Bin32* subdirectory of your SQL Anywhere installation is in your path. The SQL Anywhere PHP extension DLL requires the *Bin32* directory to be in your path.

### See also

- [“Running PHP test scripts in your web pages” on page 628](#)

## Installing the PHP extension on Linux/Solaris

To use the SQL Anywhere PHP extension on Linux or Solaris, you must copy the shared object from the SQL Anywhere installation directory and add it to your PHP installation. Optionally, you can add an entry to your PHP initialization file, *php.ini*, to load the extension, so you do not need to load it manually in each script.

### Install the PHP extension on Linux/Solaris

1. Locate the *php.ini* file of your PHP installation, and open it in a text editor. Locate the line that specifies the location of the **extension\_dir** directory. If **extension\_dir** is not set to any specific directory, it is a good idea to set it to point to an isolated directory for better system security.
2. Copy the shared object from the *lib32* or *lib64* subdirectory of your SQL Anywhere installation to the directory specified by the **extension\_dir** entry in the *php.ini* file. Your choice of shared object will depend on the version of PHP that you have installed and whether it is a 32-bit or a 64-bit version.

#### Note

If your version of PHP is more recent than the shared object provided by SQL Anywhere, try using the most recent shared object provided. Note that a version 5.2.x SQL Anywhere PHP extension will not work with a version 5.3.x PHP.

For information about which version of the shared object to use, see “[PHP extensions](#)” on page 951.

3. Add the following line to the Dynamic Extensions section of the *php.ini* file to load the SQL Anywhere PHP driver automatically. The entry must identify the shared object you copied, which is either

```
extension=php-5.x.y_sqlanywhere.so
```

or, for the thread-safe shared object,

```
extension=php-5.x.y_sqlanywhere_r.so
```

where *5.x.y* is the version number of the PHP shared object copied in the previous step.

Save and close *php.ini*.

An alternative to automatically loading the PHP driver is to load it manually in each script that requires it.

4. Before attempting to use the PHP extension, verify that your PHP execution environment is set up for SQL Anywhere. Depending on which shell you are using, you must edit the configuration script for your web server's environment and add the appropriate command to source the SQL Anywhere configuration script from the SQL Anywhere installation directory:

In this shell...	Use this command...
sh, ksh, or bash	./bin32/sa_config.sh

In this shell...	Use this command...
csh or tcsh	source /bin32/sa_config.csh

The 32-bit version of the SQL Anywhere PHP extension DLL requires the *bin32* directory to be in your path. The 64-bit version of the SQL Anywhere PHP extension DLL requires the *bin64* directory to be in your path.

The configuration file in which this line should be inserted is different for different web servers and on different Linux distributions. Here are some examples for the Apache server on the indicated distributions:

- **RedHat/Fedora/CentOS** /etc/sysconfig/httpd
- **Debian/Ubuntu** /etc/apache2/envvars

The web server must be restarted after editing its environment configuration.

#### See also

- [“Running PHP test scripts in your web pages” on page 628](#)
- [“Configuration of the SQL Anywhere PHP extension” on page 954](#)

## The PHP extension on Unix and Mac OS X

To use the SQL Anywhere PHP extension on other versions of Unix or Mac OS X, you must build the PHP extension from the source code which is installed in the *sdk/php* subdirectory of your SQL Anywhere installation.

#### See also

- [“How to build the SQL Anywhere PHP extension on Unix and Mac OS X” on page 635](#)

## Configuration of the SQL Anywhere PHP extension

The behavior of the SQL Anywhere PHP driver can be controlled by setting values in the PHP initialization file, *php.ini*. The following entries are supported:

- **extension** Causes PHP to load the SQL Anywhere PHP extension automatically each time PHP starts. Adding this entry to your PHP initialization file is optional, but if you don't add it, each script you write must start with a few lines of code that ensure that this extension is loaded. The following entry is used for Windows platforms.

```
extension=php-5.x.y_sqlanywhere.dll
```

On Linux platforms, use one of the following entries. The second entry is thread safe.

```
extension=php-5.x.y_sqlanywhere.so
```

```
extension=php-5.x.y_sqlanywhere_r.so
```

In these entries, 5.x.y identifies the PHP version.

If the SQL Anywhere extension is not always automatically loaded when PHP starts, you must prefix each script you write with the following lines of code. This code ensures that the SQL Anywhere PHP extension is loaded.

```
# Ensure that the SQL Anywhere PHP extension is loaded
if( !extension_loaded('sqlanywhere') ) {
    # Find out which version of PHP is running
    $version = phpversion();
    $extension_name = 'php-'. $version. '_sqlanywhere';
    if( strtoupper(substr(PHP_OS, 0, 3)) == 'WIN' ) {
        $extension_ext = '.dll';
    } else {
        $extension_ext = '.so';
    }
    dl( $extension_name.$extension_ext );
}
```

- **allow\_persistent** Allows persistent connections when set to On. It does not allow them when set to Off. The default value is On.

```
sqlanywhere.allow_persistent=On
```

- **max\_persistent** Sets the maximum number of persistent connections. The default value is -1, which means no limit.

```
sqlanywhere.max_persistent=-1
```

- **max\_connections** Sets the maximum number of connections that can be opened at once through the SQL Anywhere PHP extension. The default value is -1, which means no limit.

```
sqlanywhere.max_connections=-1
```

- **auto\_commit** Specifies whether the database server performs a commit operation automatically. The commit is performed immediately following the execution of each statement when set to On. When set to Off, transactions should be ended manually with either the `sasql_commit` or `sasql_rollback` functions, as appropriate. The default value is On.

```
sqlanywhere.auto_commit=On
```

- **row\_counts** Returns the exact number of rows affected by an operation when set to On or an estimate when set to Off. The default value is Off.

```
sqlanywhere.row_counts=Off
```

- **verbose\_errors** Returns verbose errors and warnings when set to On. Otherwise, you must call the `sasql_error` or `sasql_errorcode` functions to get further error information. The default value is On.

```
sqlanywhere.verbose_errors=On
```

## See also

- [“sasql\\_set\\_option” on page 659](#)

## Open Client application deployment

To deploy Open Client applications, each client computer needs the Sybase Open Client product. You must purchase the Open Client software separately from Sybase. It contains its own installation instructions.

When you use a TDS client (either Open Client or jConnect based), you have the option of sending the connection password in clear text or in encrypted form. The latter is done by performing a TDS encrypted password handshake. The handshake involves using private/public key encryption. The support for generating the RSA private/public key pair and for decrypting the encrypted password is included in a special library. The library file must be locatable by the SQL Anywhere server in its system path. For Windows, this file is called *dbrsakp12.dll*. There are both 64-bit and 32-bit versions of the DLL. On Linux and Unix environments, the file is a shared library called *libdbrsakp12.so*. On Mac OS X, the file is a shared library called *libdbrsakp12\_r.dylib*. The file is not necessary if you do not use this feature.

Connection information for Open Client clients is held in the interfaces file. For information about the interfaces file, see the Open Client documentation and “[Sybase Open Server configuration](#)” [[SQL Anywhere Server - Database Administration](#)].

## Administration tool deployment

Subject to your license agreement, you can deploy a set of administration tools including Interactive SQL, Sybase Central, and the SQL Anywhere Console utility.

The simplest way to deploy the administration tools is to use the **Deployment Wizard**.

For information about system requirements for administration tools, see <http://www.sybase.com/detail?id=1002288>.

Initialization files can simplify the deployment of the administration tools. Each of the launcher executables for the administration tools (Sybase Central, Interactive SQL, and the SQL Anywhere Console utility) can have a corresponding *.ini* file. This eliminates the need for registry entries and a fixed directory structure for the location of the JAR files. These *ini* files are located in the same directory and with the same file name as the executable file.

- ***dbconsole.ini*** This is the name of the Console utility initialization file.
- ***dbisql.ini*** This is the name of the Interactive SQL initialization file.
- ***scjview.ini*** This is the name of the Sybase Central initialization file.

The initialization file will contain the details on how to load the database administration tool. For example, the initialization file can contain the following lines:

- **JRE\_DIRECTORY=***path* This is the location of the required JRE. The **JRE\_DIRECTORY** specification is required.

- **VM\_ARGUMENTS=*any-required-VM-arguments*** VM arguments are separated by semicolons (;). Any path values that contain blanks should be enclosed in quotation marks. VM arguments can be discovered by using the `-batch` option of the administration tool and examining the corresponding batch file that is created. For example, on Windows, launching Sybase Central with `scjview -batch` at a command prompt generates `scjview.bat` and launching Interactive SQL with `dbisql -batch` generates `dbisql.bat`. The **VM\_ARGUMENTS** specification is optional.
- **JAR\_PATHS=*path1;path2;...*** A delimited list of directories which contain the JAR files for the program. They are separated by semicolons (;). The **JAR\_PATHS** specification is optional.
- **ADDITIONAL\_CLASSPATH=*path1;path2;...*** Classpath values are separated by semicolons (;). The **ADDITIONAL\_CLASSPATH** specification is optional.
- **LIBRARY\_PATHS=*path1;path2;...*** These are paths to the DLLs/shared objects. They are separated by semicolons (;). The **LIBRARY\_PATHS** specification is optional.
- **APPLICATION\_ARGUMENTS=*arg1;arg2;...*** These are any application arguments. They are separated by semicolons (;). Application arguments can be discovered by using the `-batch` option of the administration tool and examining the corresponding batch file that is created. For example, on Windows, launching Sybase Central with `scjview -batch` at a command prompt generates `scjview.bat` and launching Interactive SQL with `dbisql -batch` generates `dbisql.bat`. The **APPLICATION\_ARGUMENTS** specification is optional.

Here are the contents of a sample initialization file for Sybase Central.

```
JRE_DIRECTORY=c:\JDK\JRE160_x86
VM_ARGUMENTS=-Xmx200m
JAR_PATHS=c:\scj\jars;c:\scj\jhelp
ADDITIONAL_CLASSPATH=
LIBRARY_PATHS=c:\scj\bin
APPLICATION_ARGUMENTS=-screpository=c:\Users\Public\Documents\Sybase Central
6.1.0;-installdir=c:\scj
```

This scenario assumes that a copy of the 32-bit JRE is located in `c:\JDK\JRE160_x86`. As well, the Sybase Central executable and shared libraries (DLLs) like `jsyplib610` are stored in `c:\scj\bin`. The SQL Anywhere JAR files are stored in `c:\scj\jars`. The JavaHelp 2.0 JAR files are stored in `c:\scj\jhelp`.

#### Note

When you are deploying applications, the personal database server (dbeng12) is required for creating databases using the `dbinit` utility. It is also required if you are creating databases from Sybase Central on the local computer when no other database servers are running.

#### See also

- [“The Deployment Wizard” on page 919](#)

## Administration tool deployment on Windows

This section explains how to install Interactive SQL (dbisql), Sybase Central (including the SQL Anywhere, MobiLink, QAnywhere and UltraLite plug-ins), and the SQL Anywhere Console utility

(dbconsole) on a Windows computer without using the Deployment Wizard. It is intended for those who want to create an installer for these administration tools.

This information applies to all Windows platforms except Windows Mobile. The instructions given here are specific to version 12.0.1 and cannot be applied to earlier or later versions of the software.

**Note**

Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

**Before you begin**

Before reading this section, you should have an understanding of the Windows Registry, including the REGEDIT application. The names of the registry values are case sensitive.

**Caution**

Modifying the Windows Registry is dangerous and you do so at your own risk. It is recommended that you back up your system before modifying the registry.

The following steps are required to deploy the administration tools:

1. Decide what you want to deploy.
2. Copy the required files.
3. Register the administration tools with Windows.
4. Update the system path.
5. Register the plug-ins with Sybase Central.
6. Register the SQL Anywhere ODBC driver with Windows.
7. Register the online help files with Windows.

Each of these steps is explained in detail in the following sections.

## Step 1: Deciding what software to deploy

You can install any combination of the following software bundles:

- Interactive SQL
- Sybase Central with the SQL Anywhere plug-in
- Sybase Central with the MobiLink plug-in
- Sybase Central with the QAnywhere plug-in
- Sybase Central with the Relay Server plug-in



- Sybase Central with the UltraLite plug-in
- SQL Anywhere Console utility (dbconsole)

The following components are also required when installing any of the above software bundles:

- The SQL Anywhere ODBC Driver.
- The Java Runtime Environment (JRE) version 1.6.0. You may want to install the 32-bit version of the JRE, the 64-bit version of the JRE, or both depending on your target platform.
- The Java Access Bridge for Microsoft Windows Operating System (if you want to enable accessibility features of the administration tools on Windows platforms). Enabling this feature will permit the administration tools to be used with screen reader technologies. For more information, see <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html>.

The instructions in the following sections are structured so that you can install any (or all) of these bundles without conflicts.

## Step 2: Copying the required files

The administration tools require a specific directory structure. You are free to put the directory tree in any directory, on any drive. Throughout the following discussion, *c:\sa12* is used as the example installation folder. The software must be installed into a directory tree structure having the following layout:

Directory	Description
<i>sa12</i>	The root folder. While the following steps assume you are installing into <i>c:\sa12</i> , you are free to put the directory anywhere (for example, <i>C:\Program Files\SQLAny12</i> ).
<i>sa12\Java</i>	Holds Java program JAR files.
<i>sa12\Bin32</i>	Holds the native 32-bit Windows components used by the program, including the programs that launch the applications.
<i>sa12\Bin64</i>	Holds the native 64-bit Windows components used by the program, including the programs that launch the applications.
<i>sa12\Sun\JavaHelp-2_0</i>	The JavaHelp runtime library.
<i>sa12\Sun\jre160_x86</i>	The 32-bit Java Runtime Environment.
<i>sa12\Sun\jre160_x64</i>	The 64-bit Java Runtime Environment.

The following tables list the files required for each of the administration tools and the Sybase Central plug-ins. Make a list of the files you need, and then copy them into the directory structure outlined above.

The tables show files with the folder designation **BinXX**. There are 32-bit and 64-bit versions of these files, in the *Bin32* and *Bin64* folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders.

The tables show files with the designation **[LL]**. There are several message files each supporting a different language. If you want to install support for different languages, you have to add the resource files for these languages. For more information, see [“International message and context-sensitive help files” on page 963](#).

The administration tools require JRE 1.6.0. You should not substitute a later patch version of the JRE unless you have a specific need to do so.

If you are deploying 32-bit administration tools, then copy the 32-bit version of the JRE files from the `%SQLANY12%\Sun\jre160_x86` directory. Copy the entire *jre160\_x86* tree, including subdirectories.

If you are deploying 64-bit administration tools, then copy the 64-bit version of the JRE files from the `%SQLANY12%\Sun\jre160_x64` directory. Copy the entire *jre160\_x64* tree, including subdirectories.

### Interactive SQL

Interactive SQL (dbisql) requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbelevate12.exe
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dbisql.com
BinXX\dbisql.exe
BinXX\dbjodbc12.dll
BinXX\dblg[LL]12.dll
BinXX\dblib12.dll
BinXX\dbodbc12.dll
BinXX\jsyblib610.dll
Java\batik*.jar
Java\isql.jar
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\jsyblib610.jar
Java\pdf-transcoder.jar
Java\saip12.jar
Java\SCEditor610.jar
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar
Sun\JavaHelp-2_0\jh.jar
Sun\jre160_x86\...
Sun\jre160_x64\...
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of Interactive SQL requires the 32-bit version of the JRE files (*jre160\_x86*). The 64-bit version of Interactive SQL requires the 64-bit version of the JRE files (*jre160\_x64*).

## Sybase Central

Sybase Central (scjview) requires the following files.

```
BinXX\jsyblib610.dll
BinXX\scjview.exe
BinXX\scvw[LL]610.jar
Java\jsyblib610.jar
Java\splib.jar
Java\SCEditor610.jar
Java\sybasecentral610.jar
Sun\JavaHelp-2_0\jh.jar
Sun\jre160_x86\...
Sun\jre160_x64\...
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of Interactive SQL requires the 32-bit version of the JRE files (*jre160\_x86*). The 64-bit version of Interactive SQL requires the 64-bit version of the JRE files (*jre160\_x64*).

## Sybase Central with SQL Anywhere plug-in

The SQL Anywhere plug-in of Sybase Central requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbeng12.exe
BinXX\dbfips12.dll
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dbjodbc12.dll
BinXX\dblg[LL]12.dll
BinXX\dbl12.dll
BinXX\dbodbc12.dll
BinXX\dbput12.dll
BinXX\dbtool12.dll
BinXX\sbgse2.dll
Java\batik*.jar
Java\debugger.jar
Java\isql.jar
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\pdf-transcoder.jar
Java\saplugin.jar
Java\SQLAnywhere.jpr
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar
```

## Sybase Central with MobiLink plug-in

The MobiLink plug-in of Sybase Central requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dblg[LL]12.dll
BinXX\dbl12.dll
BinXX\dbput12.dll
BinXX\dbtool12.dll
BinXX\mljodbc12.dll
```

```
Java\isql.jar
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\mldesign.jar
Java\mlplugin.jar
Java\MobiLink.jpr
Java\stax-api-1.0.jar
Java\velocity-dep.jar
Java\velocity.jar
Java\wstx-asl-3.2.6.jar
```

### Sybase Central with QAnywhere plug-in

When deploying the QAnywhere plug-in, dbinit is required. For information about deploying database tools, see [“Database utility deployment” on page 989](#).

The QAnywhere plug-in of Sybase Central requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dblg[LL]12.dll
BinXX\dblib12.dll
BinXX\dbput12.dll
BinXX\dbtool12.dll
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\mldesign.jar
Java\mlstream.jar
Java\qaconnector.jar
Java\QAnywhere.jpr
Java\qaplugin.jar
```

### Sybase Central with Relay Server plug-in

The Relay Server plug-in of Sybase Central requires the following files.

```
Java\JComponents1200.jar
Java\jlogon.jar
Java\RelayServer.jpr
Java\rsplugin.jar
Java\rstool.jar
```

### Sybase Central with UltraLite plug-in

The UltraLite plug-in of Sybase Central requires the following files.

```
BinXX\dbdsn.exe
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dblg[LL]12.dll
BinXX\dblib12.dll
BinXX\dbput12.dll
BinXX\dbtool12.dll
BinXX\mlcecc12.dll
BinXX\mlcrsa12.dll
BinXX\mlcrsafips12.dll
BinXX\ulfips12.dll
BinXX\ulscutil12.dll
```

```

BinXX\ulutils12.dll
Java\batik*.jar
Java\isql.jar
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\js.jar
Java\pdf-transcoder.jar
Java\ulplugin.jar
Java\UltraLite.jpr
Java\xerces_2_5_0.jar
Java\xml-apis-ext.jar
Java\xml-apis.jar

```

## DBConsole

The DBConsole application requires the following files.

```

BinXX\dbconsole.exe
BinXX\dbdsn.exe
BinXX\dbelevate12.exe
BinXX\dbicu12.dll
BinXX\dbicudt12.dll
BinXX\dbjodbc12.dll
BinXX\dblg[LL]12.dll
BinXX\dblib12.dll
BinXX\dbodbc12.dll
BinXX\jsyblib610.dll
Java\DBConsole.jar
Java\JComponents1200.jar
Java\jlogon.jar
Java\jodbc4.jar
Java\jsyblib610.jar
Sun\JavaHelp-2_0\jh.jar
Sun\jre160_x86\...
Sun\jre160_x64\...

```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied. The 32-bit version of Interactive SQL requires the 32-bit version of the JRE files (*jre160\_x86*). The 64-bit version of Interactive SQL requires the 64-bit version of the JRE files (*jre160\_x64*).

## International message and context-sensitive help files

All displayed text and context-sensitive help for the administration tools is translated from English into French, German, Japanese, and Simplified Chinese. The resources for each language are held in separate files. The English files contain **en** in the file names. French files have similar names, but use **fr** instead of **en**. German file names contain **de**, Japanese file names contain **ja**, and Chinese file names contain **zh**.

If you want to install support for different languages, you have to add the message files for those other languages. There are 32-bit and 64-bit versions of these files, in the Bin32 and Bin64 folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders. The translated files are as follows:

<i>dblggen12.dll</i>	English
<i>dblgde12.dll</i>	German

<i>dblgfr12.dll</i>	French
<i>dblgja12.dll</i>	Japanese
<i>dblgzh12.dll</i>	Simplified Chinese

You must also add the context-sensitive help files for those other languages. The available translated files are as follows:

<i>scvwen610.jar</i>	English
<i>scvwde610.jar</i>	German
<i>scvwfr610.jar</i>	French
<i>scvwja610.jar</i>	Japanese
<i>scvwzh610.jar</i>	Simplified Chinese

These files are included with localized versions of SQL Anywhere.

### Accessibility components

The Java Access Bridge provides accessibility for the Java-based administration tools (you can download the Java Access Bridge for Microsoft Windows Operating System from the Java web site). Here are the install locations for the accessibility components.

<i>accessibility.properties</i>	<i>Sun\JRE160_x86\Lib</i>
<i>jaccess-1_4.jar</i>	<i>Sun\JRE160_x86\Lib\Ext</i>
<i>access-bridge.jar</i>	<i>Sun\JRE160_x86\Lib\Ext</i>
<i>JavaAccessBridge.dll</i>	a folder in the system path
<i>JAWTAccessBridge.dll</i>	a folder in the system path
<i>WindowsAccessBridge.dll</i>	a folder in the system path

Your installer must create an *accessibility.properties* file in the *JRE160\_x86\Lib* folder and it must contain an **assistive\_technologies** option as shown below. The SQL Anywhere installer creates this file.

```
#
# Load the Java Access Bridge class into the JVM
#
assistive_technologies=com.sun.java.accessibility.AccessBridge
screen_magnifier_present=true
```

## Step 3: Registering the administration tools with Windows

You must set the following registry key for the administration tools. The names of the registry values are case sensitive.

- In **HKEY\_LOCAL\_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0**
  - **Location** The fully-qualified path to the root of the installation folder (*C:\Program Files\SQL Anywhere 12* for example) containing the Sybase Central files.

You may set the following registry keys for the administration tools. The registry keys are optional and only required to be set if the OS language is different from the language that the administration tools will use. The names of the registry keys are case sensitive.

- In **HKEY\_LOCAL\_MACHINE\SOFTWARE\Sybase\Sybase Central\6.1.0**
  - **Language** The two-letter code for the language used by Sybase Central. This must be one of the following: **EN**, **DE**, **FR**, **JA**, or **ZH** for English, German, French, Japanese, and Simplified Chinese, respectively.
- In **HKEY\_LOCAL\_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0**
  - **Language** The two-letter code for the language used by SQL Anywhere and the other administration tools. This must be one of the following: **EN**, **DE**, **FR**, **JA**, or **ZH** for English, German, French, Japanese, and Simplified Chinese, respectively.

On 64-bit Windows, the equivalent registry entries for 32-bit software are under **SOFTWARE\Wow6432Node\Sybase**.

Paths should *not* end in a backslash.

Your installer can encapsulate all this information by creating a *.reg* file and then executing it. Using the example installation folder of *c:\sa12*, the following is a sample *.reg* file:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\Sybase Central\6.1.0]
"Language"="FR"

[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0]
"Location"="c:\sa12"
"Language"="FR"
```

Backslashes in file paths must be escaped by another backslash in a *.reg* file.

## Step 4: Updating the system path

To run the administration tools, the directories with *.exe* and *.dll* files must be included in the path. You must add one of the *c:\sa12\Bin32* or *c:\sa12\Bin64* directories to the system path.

On Windows, the system path is stored in the following registry key:

```
HKEY_LOCAL_MACHINE\  
SYSTEM\  
  CurrentControlSet\  
    Control\  
      Session Manager\  
        Environment\  
          Path
```

## Step 5: Creating connection profiles for Sybase Central

This step involves the configuration of Sybase Central. If you are not installing Sybase Central, you can skip it.

When Sybase Central is installed on your system, a connection profile for **SQL Anywhere 12 Demo** is created in the repository file. If you do not want to create one or more connection profiles, then you can skip this step.

The following commands were used to create the **SQL Anywhere 12 Demo** connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Name" "SQL Anywhere  
12 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/FirstTimeStart"  
"false"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Description"  
"Suitable Description"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/ProviderId"  
"sqlanywhere1200"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Provider" "SQL  
Anywhere 12"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileSettings" "DSN\eSQL^0020Anywhere^002012^0020Demo;UID  
\eDBA;PWD\xe35c624d517fb"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileName" "SQL Anywhere 12 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the repository file. Define a connection profile using Sybase Central and then look at the repository file for the corresponding lines.

Here is a portion of the repository file that was created using the process described above. Some entries have been split across multiple lines for display purposes. In the file, each entry appears on a single line:

```
# Version: 6.1.0.1154  
# Fri Feb 23 13:09:14 EST 2007  
#  
ConnectionProfiles/SQL Anywhere 12 Demo/Name=SQL Anywhere 12 Demo  
ConnectionProfiles/SQL Anywhere 12 Demo/FirstTimeStart=false  
ConnectionProfiles/SQL Anywhere 12 Demo/Description=Suitable Description  
ConnectionProfiles/SQL Anywhere 12 Demo/ProviderId=sqlanywhere1200  
ConnectionProfiles/SQL Anywhere 12 Demo/Provider=SQL Anywhere 12  
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileSettings=  
  DSN\eSQL^0020Anywhere^002012^0020Demo;  
  UID\eDBA;  
  PWD\xe35c624d517fb  
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileName=
```



```
SQL Anywhere 12 Demo
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileType=
SQL Anywhere
```

## Step 6: Registering the SQL Anywhere ODBC driver

You must install the SQL Anywhere ODBC driver before it can be used by the administration tools JDBC driver.

### See also

- [“ODBC driver configuration” on page 940](#)

## Administration tool deployment on Linux, Solaris, and Mac OS X

This section explains how to install Interactive SQL (dbisql), Sybase Central (including the SQL Anywhere, MobiLink and QAnywhere plug-ins), and the SQL Anywhere Console utility (dbconsole) on Linux, Solaris, and Mac OS X computers. It is intended for those who want to create an installer for these administration tools.

The instructions given here are specific to version 12.0.1 and may not be applicable to earlier or later versions of the software.

Note also that the dbisqlc command line utility is supported on Linux, Solaris, Mac OS X, HP-UX, and AIX. See [“dbisqlc deployment” on page 977](#).

### Note

Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Check your license agreement before considering deployment.

### Before you begin

Before you begin, you must install SQL Anywhere on one computer as a source for program files. This is the **reference installation** for your deployment.

The general steps involved are as follows:

1. Decide which programs you want to deploy.
2. Copy the required files.
3. Set environment variables.
4. Register the Sybase Central plug-ins.

Each of these steps is explained in detail in the following sections.

## Step 1: Deciding what software to deploy

You can install any combination of the following software bundles:

- Interactive SQL
- Sybase Central with the SQL Anywhere plug-in
- Sybase Central with the MobiLink plug-in
- Sybase Central with the QAnywhere plug-in
- SQL Anywhere Console utility (dbconsole)

The following components are also required when installing any of the above software bundles:

- The SQL Anywhere ODBC Driver
- The Java Runtime Environment (JRE) version 1.6.0. On Linux/Solaris, this would be the 32-bit version of the JRE. On Mac OS X, this would be the 64-bit version of the JRE.

### Note

To check your JRE version on Mac OS X, to go to the **Apple** menu, and then click **System Preferences » Software Updates**. Click **Installed Updates** for a list of updates that have been applied. If Java 1.6.0 is not in the list, go to <http://developer.apple.com/java/download/>.

The instructions in the next section are structured so that you can install any (or all) of these five bundles without conflicts.

## Step 2: Copying the required files

Your installer should copy a subset of the files that are installed by the SQL Anywhere installer. You must keep the same directory structure.

You should preserve the permissions on the files when you copy them from your reference SQL Anywhere installation. In general, all users and groups are allowed to read and execute all files.

The administration tools require JRE 1.6.0. You should not substitute a later patch version of the JRE unless you have a specific need to do so.

For Linux/Solaris, the administration tools require the 32-bit version of JRE 1.6.0. The MobiLink server requires the 64-bit version of JRE 1.6.0. For Mac OS X, the administration tools require the 64-bit version. You should not substitute a later patch version of the JRE unless you have a specific need to do so. Not all platform versions of the JRE are bundled with SQL Anywhere. The platforms that are included with SQL Anywhere support Linux on x86/x64 and Solaris SPARC. Other platforms versions must be obtained from the appropriate vendor. For example, if you are working with a Linux install, copy the entire *jre\_1.6.0\_linux\_sun\_i586* tree, including subdirectories.

If the platform that you require is included with SQL Anywhere, copy the JRE files from an installed copy of SQL Anywhere 12. Copy the entire tree, including subdirectories.

The following tables list the files required for each of the administration tools and the Sybase Central plug-ins. Make a list of the files you need, and then copy them into the directory structure outlined above.

The tables show files with the folder designation **binXX**. Depending on the platform, there are 32-bit and 64-bit versions of these files, in the *bin32* and *bin64* folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders.

The tables show files with the folder designation **libXX**. Depending on the platform, there are 32-bit and 64-bit versions of these files, in the *lib32* and *lib64* folders respectively. If you are installing 32-bit and 64-bit administration tools, then you must install both sets of files in their respective folders.

The creation of several links is required for the administration tools and Sybase Central plug-ins.

For Linux and Solaris, create symbolic links for all the shared objects that you deploy. Also, create a symbolic link in *SQLANY12/sun*. The symbolic link for Linux is *jre160\_x86* for the 32-bit JRE. The symbolic link for other systems is *jre\_160*. Here are some examples:

```
libdblib12_r.so -> $SQLANY12/lib32/libdblib12_r.so.1
jre160_x86 -> $SQLANY12/sun/jre_1.6.0_linux_sun_i586 (Linux)
jre160 -> $SQLANY12/sun/jre_1.6.0_solaris_sun_sparc (Solaris)
```

For the MobiLink plug-in on 64-bit Linux, create an additional symbolic link in *SQLANY12/sun*. The symbolic link for Linux is *jre160\_x64* for the 64-bit JRE.

```
jre160_x64 -> $SQLANY12/sun/jre_1.6.0_linux_sun_x64 (Linux)
```

For Mac OS X, shared objects have a *.dylib* extension. Symlink (symbolic link) creation is necessary for the following dylibs:

```
libdbjodbc12.jnilib -> libdbjodbc12.dylib
libdblib12_r.jnilib -> libdblib12_r.dylib
libdbput12_r.jnilib -> libdbput12_r.dylib
libmljodbc12.jnilib -> libmljodbc12.dylib
```

The tables show files with the designation **[LL]**. There are several message files each supporting a different language. If you want to install support for different languages, you have to add the resource files for these languages. For more information, see [“International message and context-sensitive help files” on page 963](#).

## Interactive SQL

Interactive SQL (dbisql) requires the following files.

```
binXX/dbisql
libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
libXX/libdbjodbc12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbodbc12_r.so (.dylib)
libXX/libdbodm12.so (.dylib)
libXX/libjsyblib610_r.so (.dylib)
```

```
java/batik*.jar
java/isql.jar
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
java/js.jar
java/jsyblib610.jar
java/pdf-transcoder.jar
java/saip12.jar
java/SCEditor610.jar
java/xerces_2_5_0.jar
java/xml-apis-ext.jar
java/xml-apis.jar
sun/JavaHelp-2_0\jh.jar
sun/jre_1.6.0_linux_sun_i586/...
sun/jre_1.6.0_solaris_sun_sparc/...
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied.

### Sybase Central

Sybase Central (scjview) requires the following files.

```
libXX/libjsyblib610_r.so (.dylib)
binXX/scjview
binXX/scvw[LL]610.jar
java/jsyblib610.jar
java/salib.jar
java/SCEditor610.jar
java/sybasecentral610.jar
sun/JavaHelp-2_0\jh.jar
sun/jre_1.6.0_linux_sun_i586/...
sun/jre_1.6.0_solaris_sun_sparc/...
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied.

### Sybase Central with SQL Anywhere plug-in

The SQL Anywhere plug-in of Sybase Central requires the following files.

```
binXX/dbeng12
libXX/libdbfips12.so (.dylib)
libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
libXX/libdbjodbc12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbodbc12_r.so (.dylib)
libXX/libdbodm12.so (.dylib)
libXX/libdbput12_r.so (.dylib)
libXX/libdbtasks12_r.so (.dylib)
libXX/libdbtool12_r.so (.dylib)
libXX/sbgse2.so
java/batik*.jar
java/debugger.jar
java/isql.jar
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
```

```

java/js.jar
java/pdf-transcoder.jar
java/sapplugin.jar
java/SQLAnywhere.jpr
java/xerces_2_5_0.jar
java/xml-apis-ext.jar
java/xml-apis.jar

```

### Sybase Central with MobiLink plug-in

The MobiLink plug-in of Sybase Central requires the following files.

```

libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbput12_r.so (.dylib)
libXX/libdbtasks12_r.so (.dylib)
libXX/libdbtool12_r.so (.dylib)
libXX/libmljodbc12.so (.dylib)
java/isql.jar
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
java/mldesign.jar
java/mlplugin.jar
java/MobiLink.jpr
java/stax-api-1.0.jar
java/velocity-dep.jar
java/velocity.jar
java/wstx-asl-3.2.6.jar

```

### Sybase Central with QAnywhere plug-in

When deploying the QAnywhere plug-in, dbinit is required. For information about deploying database tools, see [“Database utility deployment” on page 989](#).

The QAnywhere plug-in of Sybase Central requires the following files.

```

libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbput12_r.so (.dylib)
libXX/libdbtasks12_r.so (.dylib)
libXX/libdbtool12_r.so (.dylib)
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
java/mldesign.jar
java/mlstream.jar
java/qaconnector.jar
java/QAnywhere.jpr
java/qaplugin.jar

```

### Sybase Central with Relay Server plug-in

The Relay Server plug-in of Sybase Central requires the following files.

```

java/JComponents1200.jar
java/jlogon.jar

```

```
java/RelayServer.jpr
java/rsplugin.jar
java/rstool.jar
```

### Sybase Central with UltraLite plug-in

The UltraLite plug-in of Sybase Central is only available on Windows and Linux. The UltraLite plug-in of Sybase Central requires the following files.

```
libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbput12_r.so (.dylib)
libXX/libdbtasks12_r.so (.dylib)
libXX/libdbtool12_r.so (.dylib)
libXX/libmlcecc12.so (.dylib)
libXX/libmlcrsa12.so (.dylib)
libXX/libmlcrsafips12.so (.dylib)
libXX/libulfips12.so (.dylib)
libXX/libulscutil12.so (.dylib)
libXX/libulutils12.so (.dylib)
java/batik*.jar
java/isql.jar
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
java/js.jar
java/pdf-transcoder.jar
java/ulplugin.jar
java/UltraLite.jpr
java/xerces_2_5_0.jar
java/xml-apis-ext.jar
java/xml-apis.jar
```

### DBConsole

The DBConsole application requires the following files.

```
binXX/dbconsole
libXX/libdbicu12_r.so (.dylib)
libXX/libdbicudt12.so (.dylib)
libXX/libdbjodbc12.so (.dylib)
res/dblg[LL]12.res
libXX/libdblib12_r.so (.dylib)
libXX/libdbodbc12_r.so (.dylib)
libXX/libdbodm12.so (.dylib)
libXX/libjsyblib610.so (.dylib)
java/DBConsole.jar
java/JComponents1200.jar
java/jlogon.jar
java/jodbc4.jar
java/jsyblib610.jar
sun/JavaHelp-2_0\jh.jar
sun/jre_1.6.0_linux_sun_i586/...
sun/jre_1.6.0_solaris_sun_sparc/...
```

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied.

## International message and context-sensitive help files

For Linux systems only, all displayed text and context-sensitive help for the administration tools have been translated from English into German, French, Japanese, and Simplified Chinese. The resources for each language are in separate files. The English files contain **en** in the file names. German file names contain **de**, French file names contain **fr**, Japanese file names contain **ja**, and Chinese files contain **zh**.

If you want to install support for different languages, you have to add the message files for those other languages. The translated files are as follows:

<i>dblggen12.res</i>	English
<i>dblgde12_iso_1.res, dblgde12_utf8.res</i>	German (Linux only)
<i>dblgja12_eucjis.res, dblgja12_sjis.res, dblgja12_utf8.res</i>	Japanese (Linux only)
<i>dblgzh12_cp936.res, dblgzh12_eucgb.res, dblgzh12_utf8.res</i>	Simplified Chinese (Linux only)

You must also add the context-sensitive help files for those other languages. The available translated files are as follows:

<i>scvwen610.jar</i>	English
<i>scvwde610.jar</i>	German
<i>scvwfr610.jar</i>	French
<i>scvwja610.jar</i>	Japanese
<i>scvwzh610.jar</i>	Simplified Chinese

These files are included with localized versions of SQL Anywhere.

## Step 3: Setting environment variables

To run the administration tools, several environment variables must be defined or modified. This is usually done in the *sa\_config.sh* file, which is created by the SQL Anywhere installer. To use the *sa\_config.sh* file, just copy it and set `SQLANY12` to point to the deployment location.

Otherwise, to set the environment up, you must do the following:

1. Set the following environment variable:

```
SQLANY12="SQL-Anywhere-install-dir"
```

2. Set the `PATH` to include the following:

```
`${SQLANY12}/bin32
```

3. Set `LD_LIBRARY_PATH` to include the following:

For Linux:

```
$SQLANY12/jre_1.6.0_linux_sun_i586/lib/i386/client  
$SQLANY12/jre_1.6.0_linux_sun_i586/lib/i386  
$SQLANY12/jre_1.6.0_linux_sun_i586/lib/i386/native_threads
```

For Solaris:

```
$SQLANY12/jre_1.6.0_solaris_sun_sparc/lib/sparc/client  
$SQLANY12/jre_1.6.0_solaris_sun_sparc/lib/sparc  
$SQLANY12/jre_1.6.0_solaris_sun_sparc/lib/sparc/native_threads
```

On Mac OS X, the administration tools make use of a shell script stub launcher called *sa\_java\_stub\_launcher.sh*, generated at install time and placed inside the *Contents/MacOS* folder of each Java application bundle. As generated, the script sources the *System/bin64/sa\_config.sh* file to set up the environment and then runs the *JavaApplicationStub* binary, which starts the actual Java application. For deployment purposes, *sa\_java\_stub\_launcher.sh* can be modified as required to set up the environment. The name of the script can be changed by modifying the *Info.plist* file inside the Java application bundle and changing the string value of the key *CFBundleExecutable*.

## Step 4: Creating connection profiles for Sybase Central

This step involves the configuration of Sybase Central. If you are not installing Sybase Central, you can skip it.

When Sybase Central is installed on your system, a connection profile for **SQL Anywhere 12 Demo** is created in the repository file. If you do not want to create one or more connection profiles, then you can skip this step.

The following commands were used to create the **SQL Anywhere 12 Demo** connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Name" "SQL Anywhere  
12 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/FirstTimeStart"  
"false"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Description"  
"Suitable Description"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/ProviderId"  
"sqlanywhere1200"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Provider" "SQL  
Anywhere 12"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileSettings" "DSN\eSQL^0020Anywhere^002012^0020Demo;UID  
\eDBA;PWD\e35c624d517fb"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileName" "SQL Anywhere 12 Demo"  
scjview -write "ConnectionProfiles/SQL Anywhere 12 Demo/Data/  
ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the repository file. Define a connection profile using Sybase Central and then look at the repository file for the corresponding lines.

Here is a portion of the repository file that was created using the process described above. Some entries have been split across multiple lines for display purposes. In the file, each entry appears on a single line:



```
# Version: 6.1.0.1154
# Fri Feb 23 13:09:14 EST 2007
#
ConnectionProfiles/SQL Anywhere 12 Demo/Name=SQL Anywhere 12 Demo
ConnectionProfiles/SQL Anywhere 12 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 12 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 12 Demo/ProviderId=sqlanywhere1200
ConnectionProfiles/SQL Anywhere 12 Demo/Provider=SQL Anywhere 12
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileSettings=
    DSN\eSQL^0020Anywhere^002012^0020Demo;
    UID\eDBA;
    PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileName=
    SQL Anywhere 12 Demo
ConnectionProfiles/SQL Anywhere 12 Demo/Data/ConnectionProfileType=
    SQL Anywhere
```

## Administration tools configuration

You can specify which features are shown or enabled by the administration tools using the initialization file named *OEM.ini*. This file must be in the same directory as the JAR files used by the administration tools (for example, *C:\Program Files\SQL Anywhere 12\java*). If the file is not found, default values are used. Also, defaults are used for values that are missing from *OEM.ini*.

### Note

When you redeploy the administration tools, the tools cannot check for SQL Anywhere software updates. The **Check For Updates** menu items and options do not appear in redeployed versions.

Here is a sample *OEM.ini* file:

```
[errors]
# reportErrors type is boolean, default = true
reportErrors=true

[updates]
# checkForUpdates type is boolean, default = true
checkForUpdates=true

[preferences]
directory=preferences_files_directory

[dbisql]
allowPasswordsInFavorites=true
disableExecuteAll=false
# lockedPreferences is assigned a comma-separated
# list of one or more of the following option names:
#   autoCommit
#   autoRefetch
#   commitOnExit
#   disableResultsEditing
#   executeToolBarButtonSemantics
#   fastLauncherEnabled
#   maximumDisplayedRows
#   showMultipleResultSets
#   showResultsForAllStatements
lockedPreferences=showMultipleResultSets,commitOnExit
```

Any line beginning with the # character is a comment line and is ignored. The specified option names and values are case-sensitive.

The *OEM.ini* file is divided into the following sections:

### [errors]

The option set in this section applies to all administration tools.

If **reportErrors** is false, the administration tool does not present a window to the user inviting them to submit error information to iAnywhere if the software crashes. Instead, the standard window appears.

### [updates]

The option set in this section applies to all administration tools.

If **checkForUpdates** is false, the administration tool does not check for SQL Anywhere software updates automatically, nor does it give the user the option to do so at their discretion.

### [preferences]

The option set in this section applies to all administration tools.

You can set the **directory** option to specify the directory used by the administration tools to save user-specific configuration files. These files contain information related to the administration tools' settings and history. For example, they may contain Interactive SQL statement history, recently opened files, or saved window positions.

You must specify a fully-qualified directory name (such as *c:\work\prefs*) that does not end with a path separator (backslash on Windows; forward slash on Linux, Unix, and OS X).

On Windows, the default setting for the user preferences directory is *%userprofile%\sybase*. This setting permits multiple users on a single system to each have their own directory for the recording of preferences. Overriding this setting in the *OEM.ini* file disables this capability.

### [dbisql]

The options set in this section apply only to Interactive SQL.

- **allowPasswordsInFavorites** If **allowPasswordsInFavorites** is false, Interactive SQL removes the **Save The Connection Password** checkbox from the **Add To Favorites** window. The default setting is true, which means the checkbox is present. See [“Saving SQL script files and connections in a favorites list” \[SQL Anywhere Server - Database Administration\]](#).
- **disableExecuteAll** If **disableExecuteAll** is true, then the **SQL » Execute** menu item and the F5 accelerator key are disabled in Interactive SQL. If the Execute toolbar button is configured for **Execute**, then it is disabled also. Therefore, you might want to set the Execute toolbar button to **Execute Selection** in Interactive SQL, and then set the `executeToolBarButtonSemantics` option in the *OEM.ini* file to prevent users from changing the Execute toolbar button. See [“Configuring the Execute Statements toolbar button” \[SQL Anywhere Server - Database Administration\]](#).
- **lockedPreferences** You can lock the settings of Interactive SQL options so that users cannot change them. The option names are case sensitive. The following is an example:

```
[dbisql]
lockedPreferences=autoCommit
```

You can prevent users from changing the following Interactive SQL option settings:

- **autoCommit** Prevents users from customizing the **Commit After Every Statement** option. See “[auto\\_commit option \[Interactive SQL\]](#)” [*SQL Anywhere Server - Database Administration*].
- **autoRefetch** Prevents users from customizing the **Automatically Refetch Results** option. See “[auto\\_refetch option \[Interactive SQL\]](#)” [*SQL Anywhere Server - Database Administration*].
- **commitOnExit** Prevents users from customizing the **Commit On Exit Or Disconnect** option. See “[commit\\_on\\_exit option \[Interactive SQL\]](#)” [*SQL Anywhere Server - Database Administration*].
- **disableResultsEditing** Prevents users from customizing the **Disable Editing** option.
- **executeToolBarButtonSemantics** Prevents users from customizing the behavior of the **Execute** toolbar button. See “[Configuring the Execute Statements toolbar button](#)” [*SQL Anywhere Server - Database Administration*].
- **fastLauncherEnabled** Prevents users from customizing the fast launcher option. See “[Fast launcher option](#)” [*SQL Anywhere Server - Database Administration*].
- **maximumDisplayedRows** Prevents users from customizing the **Maximum Number Of Rows To Display** option. See “[isql\\_maximum\\_displayed\\_rows option \[Interactive SQL\]](#)” [*SQL Anywhere Server - Database Administration*].
- **showMultipleResultSets** Prevents users from customizing the **Show Only The First Result Set** or **Show All Result Sets** options. This setting also prevents the user from setting the `isql_show_multiple_result_sets` option. See “[isql\\_show\\_multiple\\_result\\_sets option \[Interactive SQL\]](#)” [*SQL Anywhere Server - Database Administration*].
- **showResultsForAllStatements** Prevents users from customizing the **Show Results From The Last Statement** or **Show Results From Each Statement** options. See “[Returning multiple result sets from procedures](#)” [*SQL Anywhere Server - SQL Usage*].

## dbisqlc deployment

If your deployed application requires a query execution or scripting tool and is running on computers with limited resources, you may want to deploy the dbisqlc executable instead of Interactive SQL (dbisql). However, dbisqlc is deprecated, and no new features are being added to it. Also, dbisqlc does not contain all the features of Interactive SQL and compatibility between the two is not guaranteed.

The dbisqlc executable requires the standard embedded SQL client-side libraries.

### See also

- “dbisqlc utility (deprecated)” [[SQL Anywhere Server - Database Administration](#)]
- “Interactive SQL utility (dbisql)” [[SQL Anywhere Server - Database Administration](#)]

## Documentation deployment

The discussion below uses the designation [LL]. Documentation is available in different languages. Replace the [LL] with the language code for the documentation you are deploying (for example, **en**, **de**, **jp**, and so on).

For Windows, there are two options available for documentation. You can either use DCX help, in which case no help files need to be deployed at all (you get help from the [dcx.sybase.com](#) web site) or you can deploy HTML-based help.

HTML-based help documentation is installed to the `%SQLANY12%\Documentation` directory tree.

```
Documentation\sqlanywhere_[LL]12.map
Documentation\[LL]\htmlhelp\dbadmin12.chm
Documentation\[LL]\htmlhelp\dbprogramming12.chm
Documentation\[LL]\htmlhelp\dbreference12.chm
Documentation\[LL]\htmlhelp\dbspatial12.chm
Documentation\[LL]\htmlhelp\dbusage12.chm
Documentation\[LL]\htmlhelp\mlclient12.chm
Documentation\[LL]\htmlhelp\mlserver12.chm
Documentation\[LL]\htmlhelp\mlsisync12.chm
Documentation\[LL]\htmlhelp\mlstart12.chm
Documentation\[LL]\htmlhelp\qanywhere12.chm
Documentation\[LL]\htmlhelp\relayserver12.chm
Documentation\[LL]\htmlhelp\sachanges12.chm
Documentation\[LL]\htmlhelp\sacshelp12.chm
Documentation\[LL]\htmlhelp\saerrors12.chm
Documentation\[LL]\htmlhelp\saintro12.chm
Documentation\[LL]\htmlhelp\sausinghelp12.chm
Documentation\[LL]\htmlhelp\sqlanywhere_[LL]12.chm
Documentation\[LL]\htmlhelp\sqlremote12.chm
Documentation\[LL]\htmlhelp\uladmin12.chm
Documentation\[LL]\htmlhelp\ulc12.chm
Documentation\[LL]\htmlhelp\uldotnet12.chm
Documentation\[LL]\htmlhelp\ulj12.chm
Documentation\[LL]\htmlhelp\ulmbus12.chm
```

For Linux, Unix, and Mac OS X, there are two options available for documentation. You can either use DCX help, in which case no help files need to be deployed at all (you get help from the [dcx.sybase.com](#) web site) or you can deploy Eclipse help.

Eclipse help can be installed locally with the documentation installer (`sa12_doc_[LL]_linux_x86+x64.[version].[build].tar.gz`). There are versions available for different languages.

For example, if you download `sa12_doc_en_linux_x86+x64.1200.2406.tar.gz` then the following command can be used to install the documentation.

```
/setup -ss -sqlany-dir $SQLANY12
```

This silently installs the help to the location specified by the SQLANY12 environment variable.

## Database server deployment

You can deploy a database server by making the SQL Anywhere installer available to your end users. By selecting the proper option, each end user is guaranteed of getting the files they need.

The simplest way to deploy a personal database server or a network database server is to use the **Deployment Wizard**. For more information, see [“The Deployment Wizard” on page 919](#).

To run a database server, you need to install a set of files. The files are listed in the following table. All redistribution of these files is governed by the terms of your license agreement. You must confirm whether you have the right to redistribute the database server files before doing so.

<b>Windows</b>	<b>Linux / Unix</b>	<b>Mac OS X</b>
<i>dbeng12.exe</i>	<i>dbeng12</i>	<i>dbeng12</i>
<i>dbeng12.lic</i>	<i>dbeng12.lic</i>	<i>dbeng12.lic</i>
<i>dbsrv12.exe</i>	<i>dbsrv12</i>	<i>dbsrv12</i>
<i>dbsrv12.lic</i>	<i>dbsrv12.lic</i>	<i>dbsrv12.lic</i>
<i>dbserv12.dll</i>	<i>libdbserv12_r.so, libdb-tasks12_r.so</i>	<i>libdbserv12_r.dylib, libdb-tasks12_r.dylib</i>
<i>dbscript12.dll</i>	<i>libdbscript12_r.so</i>	<i>libdbscript12_r.dylib</i>
<i>dblg[LL]12.dll</i>	<i>dblg[LL]12.res</i>	<i>dblggen12.res</i>
<i>dbghelp.dll</i>	N/A	N/A
<i>dbctrs12.dll</i>	N/A	N/A
<i>dbextf.dll</i> <sup>1</sup>	<i>libdbextf.so</i> <sup>1</sup>	<i>libdbextf.dylib</i> <sup>1</sup>
<i>dbicu12.dll</i> <sup>2</sup>	<i>libdbicu12_r.so</i> <sup>2</sup>	<i>libdbicu12_r.dylib</i> <sup>2</sup>
<i>dbicudt12.dll</i> <sup>2, 3</sup>	<i>libdbicudt12.so</i> <sup>2</sup>	<i>libdbicudt12.dylib</i> <sup>2</sup>
<i>sqlany.cvf</i>	<i>sqlany.cvf</i>	<i>sqlany.cvf</i>
<i>dbrsakp12.dll</i> <sup>4</sup>	<i>libdbrsakp12_r.so</i> <sup>4</sup>	<i>libdbrsakp12_r.dylib</i> <sup>4</sup>
<i>dbodbc12.dll</i> <sup>5</sup>	<i>libdbodbc12.so</i> <sup>5</sup>	<i>libdbodbc12.dylib</i> <sup>5</sup>

Windows	Linux / Unix	Mac OS X
<i>dbjodbc12.dll</i> <sup>5</sup>	<i>libdbjodbc12.so</i> <sup>5</sup>	<i>libdbjodbc12.dylib</i> <sup>5</sup>
N/A	<i>libdbodbc12_n.so</i> <sup>5</sup>	<i>libdbodbc12_n.dylib</i> <sup>5</sup>
N/A	<i>libdbodbc12_r.so</i> <sup>5</sup>	<i>libdbodbc12_r.dylib</i> <sup>5</sup>
<i>dbjdbc12.dll</i> <sup>6</sup>	<i>libdbjdbc12.so</i> <sup>6</sup>	<i>libdbjdbc12.dylib</i> <sup>6</sup>
<i>java/jconn3.jar</i> <sup>6</sup>	<i>java/jconn3.jar</i> <sup>6</sup>	<i>java/jconn3.jar</i> <sup>6</sup>
<i>java\sajdbc.jar</i> <sup>6, 10</sup>	<i>java\sajdbc.jar</i> <sup>6, 10</sup>	<i>java\sajdbc.jar</i> <sup>6, 10</sup>
<i>java\sajdbc4.jar</i> <sup>6</sup>	<i>java\sajdbc4.jar</i> <sup>6</sup>	<i>java\sajdbc4.jar</i> <sup>6</sup>
<i>java\sajvm.jar</i> <sup>6</sup>	<i>java\sajvm.jar</i> <sup>6</sup>	<i>java\sajvm.jar</i> <sup>6</sup>
<i>java\cis.zip</i> <sup>7</sup>	<i>java\cis.zip</i> <sup>7</sup>	<i>java\cis.zip</i> <sup>7</sup>
<i>dbcis12.dll</i> <sup>8</sup>	<i>libdbcis12.so</i> <sup>8</sup>	<i>libdbcis12.dylib</i> <sup>8</sup>
<i>libsybbr.dll</i> <sup>9</sup>	<i>libsybbr.so</i> <sup>9</sup>	<i>libsybbr.dylib</i> <sup>9</sup>

<sup>1</sup> Required only if using system extended stored procedures and functions (xp\_\*).

<sup>2</sup> Required only if the database character set is multi-byte or if the UCA collation sequence is used.

<sup>3</sup> On Windows Mobile, the file name to deploy is called *dbicudt12.dat*.

<sup>4</sup> Required only for encrypted TDS connections.

<sup>5</sup> Required only if using Java in the database with version 10 databases.

<sup>6</sup> Required only if using Java in the database.

<sup>7</sup> Required only if using Java in the database and remote data access.

<sup>8</sup> Required only if using remote data access.

<sup>9</sup> Required only for archive backups.

<sup>10</sup> Required only for JDBC 3.0.

## Notes

- Depending on your situation, you should choose whether to deploy the personal database server (dbeng12) or the network database server (dbsrv12).
- You must include the separate corresponding license file (*dbeng12.lic* or *dbsrv12.lic*) when deploying a database server. The license files are located in the same directory as the server executables.

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **jp**, and so on).
- The Java VM jar file (*sajvm.jar*) is required only if the database server is to use the Java in the Database functionality.
- The table does not include files needed to run utilities such as dbbackup.  
For information about deploying utilities, see [“Administration tool deployment” on page 956](#).
- The database server **xd** option is useful in deployed applications since it prevents the database server from becoming the default database server. Connections are accepted only when the server name is included in the connection string.  
For information about the **xd** option, see [“-xd dbeng12/dbsrv12 server option” \[SQL Anywhere Server - Database Administration\]](#).  
For information about specifying the server name, see [“ServerName \(Server\) connection parameter” \[SQL Anywhere Server - Database Administration\]](#).

## Windows Registry entries

To improve robustness across power failures on systems using certain Intel storage drivers, specific registry entries must be set. To determine if this step is required, first check that the parent key exists and then add a parameter value to the key (the steps are described below). Failure to set this parameter can result in lost data and corrupted databases in the event of a power failure.

### Improve robustness on Intel storage drivers

1. Check that the following registry entry exists.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastor\Parameters\
```

2. If it does then add a REG\_DWORD value named **EnableFlush** within this key and assign it the data value 1. Here is a sample registry change file.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastor\Parameters]
"EnableFlush"=dword:00000001
```

3. Check that the following registry entry exists.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\iastorv\Parameters\
```

4. If it does then add a REG\_DWORD value named **EnableFlush** within this key and assign it the data value 1. Here is a sample registry change file.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iastorv\Parameters]
"EnableFlush"=dword:00000001
```

To ensure that messages written by the server to the Event Log on Windows are formatted correctly, you must create a registry key.

## Format Event Log messages properly

1. Create the following registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 12.0
```

2. Within this key, add a REG\_SZ value named **EventMessageFile** and assign it the data value of the fully qualified location of *dblgen12.dll*, for example, *C:\Program Files\SQL Anywhere 12\Bin32\dblgen12.dll*. The English language DLL, *dblgen12.dll*, can be specified regardless of the deployment language. Here is a sample registry change file.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 12.0]
"EventMessageFile"="c:\sa12\bin32\dblgen12.dll"
```

For the 64-bit version of the server, the registry key is **SQLANY64 12.0**.

3. To ensure that messages written by MESSAGE...TO EVENT LOG statements to the Event Log on Windows are formatted correctly, create the following registry key.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 12.0 Admin
```

4. Within this key, add a REG\_SZ value named **EventMessageFile** and assign it the data value of the fully qualified location of *dblgen12.dll*, for example, *C:\Program Files\SQL Anywhere 12\Bin32\dblgen12.dll*. The English language DLL, *dblgen12.dll*, can be specified regardless of the deployment language. Here is a sample registry change file.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application
\SQLANY 12.0 Admin]
"EventMessageFile"="c:\sa12\bin32\dblgen12.dll"
```

For the 64-bit version of the server, the registry key is **SQLANY64 12.0 Admin**.

5. You can suppress Windows event log entries by setting up a registry key. The registry key is:

```
Software\Sybase\SQL Anywhere\12.0\EventLogMask
```

and it can be placed in either the HKEY\_CURRENT\_USER or HKEY\_LOCAL\_MACHINE hive. To control event log entries, create a REG\_DWORD value named **EventLogMask** and assign it a bit mask containing the internal bit values for the different Windows event types. The three types supported by the SQL Anywhere database server are:

```
EVENTLOG_ERROR_TYPE          0x0001
EVENTLOG_WARNING_TYPE        0x0002
EVENTLOG_INFORMATION_TYPE    0x0004
```

For example, if the **EventLogMask** key is set to zero, no messages appear at all. A better setting would be 1, so that informational and warning messages don't appear, but errors do. The default setting (no entry present) is for all message types to appear. Here is a sample registry change file.



```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0]
"EventLogMask"=dword:00000007
```

## Mac OS X considerations

On Mac OS X, DBLauncher and SyncConsole will use the User Defaults system to locate the SQL Anywhere installation. The SQL Anywhere installer writes the installation location to the user defaults repository of the user performing the installation. Other users will be prompted for the installation directory when DBLauncher is first used. Also, the Preferences panel can be used to change this setting.

## Database deployment

You deploy a database file by installing the database file onto your end-user's disk.

As long as the database server shuts down cleanly, you do not need to deploy a transaction log file with your database file. When your end user starts running the database, a new transaction log is created.

For SQL Remote applications, the database should be created in a properly synchronized state, in which case no transaction log is needed. You can use the Extraction utility for this purpose.

### See also

- [“Remote database extraction” \[SQL Remote\]](#)

## International considerations

When you are deploying a database worldwide, you should consider the locales in which the database is to be used. Different locales may have different sort orders or text comparison rules. For example, the database that you are going to deploy maybe have been created using the 1252LATIN1 collation and this may not be suitable for some of the environments in which it is to be used.

Since the collation of a database cannot be changed after it has been created, you might consider creating the database during the install phase and then populating the database with the schema and data that you want it to have. The database can be created during the install either by using the dbinit utility, or by starting the database server with the utility database and issuing a CREATE DATABASE statement. Then, you can use SQL statements to create the schema and do whatever else is necessary to set up your initial database.

If you decide to use the UCA collation, you can specify additional collation tailoring options for finer control over the sorting and comparing of characters using the dbinit utility or the CREATE DATABASE statement. These options take the form of *keyword=value* pairs, assembled in parentheses, following the collation name. Using the CREATE DATABASE statement, for example, a collation tailoring can be specified using syntax like the following:

```
CHAR COLLATION 'UCA( locale=es;case=respect;accent=respect )'
```

As an alternative, you could create multiple database templates, one for each of the locales in which the database is to be used. This may be suitable if the set of locales you are deploying the database to is relatively small. You can have the installer choose which database to install.

### See also

- “CREATE DATABASE statement” [[SQL Anywhere Server - SQL Reference](#)]
- “Initialization utility (dbinit)” [[SQL Anywhere Server - Database Administration](#)]
- “Collation considerations” [[SQL Anywhere Server - Database Administration](#)]

## Database deployment on read-only media

You can distribute databases on read-only media, such as a CD-ROM, as long as you run them in read-only mode.

For more information about running databases in read-only mode, see “-r dbeng12/dbsrv12 server option” [[SQL Anywhere Server - Database Administration](#)].

If you need to make changes to the database, you must copy the database from the CD-ROM to a location where it can be modified, such as a hard drive.

## Registering DLLs on Windows

Some DLL files require registration if deployed for use with SQL Anywhere. Note that for Windows Vista or later versions of Windows, you must include the SQL Anywhere elevated operations agent (*dbelevate12.exe*) which supports the privilege elevation required when DLLs are registered or unregistered.

There are many ways you can register these DLLs, including in an install script or using the regsvr32 utility on Windows or the regsvrce utility on Windows Mobile. You could also include a command, like the one in the following procedure, in a batch file.

### Register a DLL

1. Open a command prompt.
2. Change to the directory where the DLL is installed.
3. Enter the following command to register the DLL (in this example, the ODBC driver is registered):

```
regsvr32 dbodbc12.dll
```

The following table lists the DLLs that require registration if deployed on Windows:

File	Description
<i>dbctrs12.dll</i>	The SQL Anywhere Performance Monitor counters.
<i>dbodbc12.dll</i>	The SQL Anywhere ODBC driver.
<i>dboledb12.dll</i>	The SQL Anywhere OLE DB provider.
<i>dboledba12.dll</i>	The SQL Anywhere OLE DB provider schema assist module (Windows only).
<i>msxml4.dll</i>	The Microsoft XML Parser (used by UltraLite 32-bit only for unloading UltraLite version 9.0 databases).

## External environment support deployment

The following tables summarize the components that must be deployed to support external calls in SQL Anywhere.

### ESQL/ODBC external calls

Component	Windows	Linux / Unix	Mac OS X
ESQL and ODBC launcher	<i>dbexternc12.exe</i>	<i>dbexternc12</i>	<i>dbexternc12</i>
Bridge	<i>dbxtenv12.dll</i>	<i>libdbx- tenv12_r.so</i>	<i>libdbx- tenv12_r.dylib</i>
SQL Anywhere C API runtime	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>

For additional files required by embedded SQL applications, see [“Embedded SQL client deployment” on page 945](#).

For additional files required by ODBC applications, see [“ODBC client deployment” on page 936](#).

### Java external calls

Component	Windows	Linux / Unix	Mac OS X
Java installation (third-party)	<i>java.exe</i>	<i>java</i>	<i>java</i>
Launcher	<i>sajvm.jar</i>	<i>sajvm.jar</i>	<i>sajvm.jar</i>

Component	Windows	Linux / Unix	Mac OS X
SQL Anywhere JDBC driver (server-side calls)	<i>dbjdbc12.dll</i>	<i>libdbjdbc12.so</i>	<i>libdbjdbc12.dylib</i>

For additional files required by JDBC applications, see [“JDBC client deployment” on page 947](#).

### .NET CLR external calls

Component	Windows	Linux / Unix	Mac OS X
.NET 3.5 or later	(from Microsoft)	N/A	N/A
SQL Anywhere .NET 3.5 or 4.0 provider	(from Sybase)	N/A	N/A
.NET CLR bridge	<i>dbextclr12.exe</i>	N/A	N/A
Bridge	<i>dbextenv12.dll</i>	N/A	N/A
.NET CLR support	<i>dbclrenv12.dll</i>	N/A	N/A
DBLIB	<i>dblib12.dll</i>	N/A	N/A

### Perl external calls

Component	Windows	Linux / Unix	Mac OS X
Perl installation (third-party)	<i>perl.exe</i>	<i>perl</i>	<i>perl</i>
Perl launcher	<i>perlenv.pl</i>	<i>perlenv.pl</i>	<i>perlenv.pl</i>
Bridge	<i>dbextenv12.dll</i>	<i>libdbextenv12_r.so</i>	<i>libdbextenv12_r.dylib</i>
SQL Anywhere C API runtime	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>

For additional files required by Perl applications, also [“DBD::SQLAnywhere” on page 611](#).

### PHP external calls

Component	Windows	Linux / Unix	Mac OS X
PHP installation (third-party)	<i>php.exe</i>	<i>php</i>	<i>php</i>

Component	Windows	Linux / Unix	Mac OS X
PHP launcher	<i>phpenv.php</i>	<i>phpenv.php</i>	<i>phpenv.php</i>
Bridge	<i>dbextenv12.dll</i>	<i>libdbextenv12_r.so</i>	<i>libdbextenv12_r.dylib</i>
PHP 5.1.x external calls	<i>php-5.1.[1-6]_sqlanywhere_extenv12.dll</i>	<i>php-5.1.[1-6]_sqlanywhere_extenv12_r.so</i> or build from source code	Build from source code
PHP 5.2.x external calls	<i>php-5.2.[0-11]_sqlanywhere_extenv12.dll</i>	<i>php-5.2.[0-11]_sqlanywhere_extenv12_r.so</i> or build from source code	Build from source code
PHP 5.3.x external calls	<i>php-5.3.[0-5]_sqlanywhere_extenv12.dll</i>	<i>php-5.3.[0-5]_sqlanywhere_extenv12_r.so</i> or build from source code	Build from source code
SQL Anywhere C API runtime	<i>dbcapi.dll</i>	<i>libdbcapi_r.so</i>	<i>libdbcapi_r.dylib</i>
DBLIB (threaded)	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>
Thread support library	N/A	<i>libdbtasks12_r.so</i>	<i>libdbtasks12_r.dylib</i>

For additional files required by PHP applications, see [“SQL Anywhere PHP extension”](#) on page 627.

### See also

- [“The ESQLE and ODBC external environments”](#) on page 587
- [“The Java external environment”](#) on page 596
- [“The CLR external environment”](#) on page 584
- [“The PERL external environment”](#) on page 601
- [“The PHP external environment”](#) on page 605

## Security deployment

The following table summarizes the components that support security features in SQL Anywhere.

Security option	Security type	Included in module	Licensable
Database encryption	AES	<i>dbserv12.dll</i> <i>libdserv12_r.so</i>	included <sup>1</sup>
Database encryption	FIPS-certified AES	<i>dbfips12.dll</i>	separately licensed <sup>2</sup>

Security option	Security type	Included in module	Licensable
Transport layer security	RSA	<i>dbrsa12.dll</i> <i>libdbrsa12.so</i> <i>libdbrsa12.dylib</i> <i>libdbrsa12_r.dylib</i>	included <sup>1</sup>
Transport layer security	FIPS-certified RSA	<i>dbfips12.dll, sbgse2.dll</i>	separately licensed <sup>2</sup>
Transport layer security	ECC	<i>dbecc12.dll</i> <i>libdbecc12.so</i>	separately licensed <sup>2</sup>

<sup>1</sup> AES and RSA encryption are included with SQL Anywhere and do not require a separate license, but these libraries are not FIPS-certified.

<sup>2</sup> The software for encryption using ECC or FIPS-certified technology must be ordered separately.

## Embedded database application deployment

This section provides information on deploying embedded database applications, where the application and the database both reside on the same computer.

An embedded database application includes the following:

- **Client application** This includes the SQL Anywhere client requirements.
- **Database server** The SQL Anywhere personal database server.
- **SQL Remote** If your application uses SQL Remote replication, you must deploy the SQL Remote Message Agent.
- **The database** You must deploy a database file holding the data the application uses.

### See also

- [“Database server deployment” on page 979](#)
- [“Requirements for deploying client applications” on page 927](#)

## Personal server deployment

When you deploy an application that uses the personal server, you need to deploy both the client application components and the database server components.

The language resource library (*dblggen12.dll*) is shared between the client and the server. You need only one copy of this file.

It is recommended that you follow the SQL Anywhere installation behavior, and install the client and server files in the same directory.

Remember to provide the Java zip files and the Java DLL if your application takes advantage of Java in the Database.

## Database utility deployment

If you need to deploy database utilities (such as *dbbackup*) along with your application, then you need the utility executable together with the following additional files:

Description	Windows	Linux / Unix	Mac OS X
Database tools library	<i>dbtool12.dll</i>	<i>libdbtool12_r.so, libdbtasks12_r.so</i>	<i>libdbtool12_r.dylib, libdbtasks12_r.dylib</i>
Interface Library	<i>dblib12.dll</i>	<i>libdblib12_r.so</i>	<i>libdblib12_r.dylib</i>
Language resource library	<i>dblg[LL]12.dll</i>	<i>dblg[LL]12.res</i>	<i>dblggen12.res</i>
Connect window	<i>dbcon12.dll</i>		
Version 9 or earlier physical store library	<i>dboftsp.dll</i>	<i>libdboftsp_r.so</i>	N/A

### Notes

- A language resource library file should also be included. The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace [LL] with the language code (for example, **en**, **de**, **jp**, and so on).
- For non-multithreaded applications on Linux and Unix, you can use *libdbtasks12.so* and *libdblib12.so*.
- For non-multithreaded applications on Mac OS X, you can use *libdbtasks12.dylib* and *libdblib12.dylib*.
- For Windows, the **ODBC Configuration for SQL Anywhere** and **Connect to SQL Anywhere** window support code (*dbcon12.dll*) is needed if your end users will create their own data sources, if they need to enter user IDs and passwords when connecting to the database, or if they need to display the Connect window for any other purpose.
- The version 9 or earlier physical store library is required by some utilities (*dblog*, *dbtran*, *dberase*) to access log files created by versions of the software earlier than 10.0.0. If you are not deploying these utilities, then you do not require this library.

- If you require support for unloading version 9.0 or earlier databases, then you need the database unload utility, dbunload, together with the files described in [“Unload support deployment for version 9 and earlier databases”](#).
- The personal database server (dbeng12) is required for creating databases using the dbinit utility. It is also required if you are creating databases from Sybase Central on the local computer when no other database servers are running.

**See also**

- [“Database server deployment” on page 979](#)

## Unload support deployment for version 9 and earlier databases

If, in your application, you need to the ability to convert version 9.0 or older databases to the current format, then you need the database unload utility, dbunload, together with the following additional files:

Description	Windows	Linux / Unix	Mac OS X
Unload support for pre-10.0 databases	<i>dbunlspt.exe</i>	<i>dbunlspt</i>	<i>dbunlspt</i>
Message resource library	<i>dbus[LL].dll</i>	<i>dbus[LL].res</i>	<i>dbus[LL].res</i>
Unload script file	<i>optdeflt.sql</i>	<i>optdeflt.sql</i>	<i>optdeflt.sql</i>
Unload script file	<i>opttemp.sql</i>	<i>opttemp.sql</i>	<i>opttemp.sql</i>
Unload script file	<i>unloadold.sql</i>	<i>unloadold.sql</i>	<i>unloadold.sql</i>

The table above shows files with the designation [LL]. There are several message files each supporting a different language. If you want to install support for different languages, you have to add the resource files for these languages. The message files are as follows.

**Windows message files**

<i>dbusde.dll</i>	German
<i>dbusen.dll</i>	English
<i>dbuses.dll</i>	Spanish
<i>dbusfr.dll</i>	French
<i>dbusit.dll</i>	Italian



<i>dbusja.dll</i>	Japanese
<i>dbusko.dll</i>	Korean
<i>dbuslt.dll</i>	Lithuanian
<i>dbuspl.dll</i>	Polish
<i>dbuspt.dll</i>	Portuguese
<i>dbusru.dll</i>	Russian
<i>dbustw.dll</i>	Traditional Chinese
<i>dbusuk.dll</i>	Ukrainian
<i>dbuszh.dll</i>	Simplified Chinese

### Linux message files

<i>dbusde_iso_1.res, dbusde_utf8.res, dbusen.res</i>	German
<i>dbusen.res</i>	English
<i>dbusja_eucjis.res, dbusja_sjis.res, dbusja_utf8.res</i>	Japanese
<i>dbuszh_cp936.res, dbuszh_eucgb.res, dbuszh_utf8.res</i>	Chinese

These files are included with localized versions of SQL Anywhere.

In addition to these files, you also need the files described in [“Database utility deployment” on page 989](#).

## SQL Remote deployment

If you are deploying the SQL Remote Message Agent, you need to include the following files:

<b>Description</b>	<b>Windows</b>	<b>Linux / Solaris</b>	<b>Mac OS X</b>
Message Agent	<i>dbremote.exe</i>	<i>dbremote</i>	<i>dbremote</i>
Encoding/ decoding li- brary	<i>dbencod12.dll</i>	<i>libdbencod12_r.so.1</i>	<i>libdbencod12_r.dylib</i>

<b>Description</b>	<b>Windows</b>	<b>Linux / Solaris</b>	<b>Mac OS X</b>
FILE message link library <sup>1</sup>	<i>dbfile12.dll</i>	<i>libdbfile12_r.so.1</i>	<i>libdbfile12_r.dylib</i>
FTP message link library <sup>1</sup>	<i>dbftp12.dll</i>	<i>libdbftp12_r.so.1</i>	<i>libdbftp12_r.dylib</i>
Language resource library	<i>dblg[LL]12.dll</i>	<i>dblg[LL]12.res</i>	<i>dblg[LL]12.res</i>
Interface Library	<i>dblib12.dll</i>	<i>libdblib12_r.so.1</i>	<i>libdblib12_r.dylib</i>
SMTP message link library <sup>1</sup>	<i>dbsmtp12.dll</i>	<i>libdbsmtp12_r.so.1</i>	<i>libdbsmtp12_r.dylib</i>
Database tools library	<i>dbtool12.dll</i>	<i>libdbtool12_r.so.1</i>	<i>libdbtool12_r.dylib</i>
Thread support library	N/A	<i>libdbtasks12_r.so.1</i>	<i>libdbtasks12_r.dylib</i>

<sup>1</sup> Only deploy the library for the message link you are using.

A language resource library file should also be included. The table above shows files with the designation **[LL]**. There are several message files each supporting a different language. If you want to install support for different languages, you have to include the resource files for these languages. Replace **[LL]** with the language code (for example, **en**, **de**, **jp**, and so on).

It is recommended that you follow the SQL Anywhere installation behavior, and install the SQL Remote files in the same directory as the SQL Anywhere files.

---

# Index

## Symbols

- d option
    - SQL preprocessor utility (sqlpp), 437
  - e option
    - SQL preprocessor utility (sqlpp), 437
  - gn option
    - threads, 398
  - h option
    - SQL preprocessor utility (sqlpp), 437
  - install option
    - SetupVSPackage, 927
  - k option
    - SQL preprocessor utility (sqlpp), 437
  - m option
    - SQL preprocessor utility (sqlpp), 437
  - n option
    - SQL preprocessor utility (sqlpp), 437
  - o option
    - SQL preprocessor utility (sqlpp), 437
  - q option
    - SQL preprocessor utility (sqlpp), 437
  - r option
    - SQL preprocessor utility (sqlpp), 437
  - s option
    - SQL preprocessor utility (sqlpp), 437
  - salocation option
    - SetupVSPackage, 927
  - u option
    - SQL preprocessor utility (sqlpp), 437
  - uninstall option
    - SetupVSPackage, 929
  - w option
    - SQL preprocessor utility (sqlpp), 437
  - x option
    - SQL preprocessor utility (sqlpp), 437
  - z option
    - SQL preprocessor utility (sqlpp), 437
  - .NET
    - data control, 84
    - deploying, 927
    - using the SQL Anywhere .NET Data Provider, 39
  - .NET API
    - about, 39
  - .NET Data Provider
    - about, 39
    - accessing data, 46
    - adding a reference, 42
    - connecting to a database, 43
    - connection pooling, 45
    - dbdata.dll, 71
    - deleting data, 46
    - deploying, 69
    - Entity Framework support, 64
    - error handling, 63
    - executing stored procedures, 61
    - features, 40
    - files required for deployment, 70
    - iAnywhere.Data.SQLAnywhere provider, 40
    - inserting data, 46
    - obtaining time values, 60
    - POOLING option, 45
    - referencing the provider classes in your source code, 42
    - registering, 71
    - running the sample projects, 41
    - supported languages, 39
    - system requirements, 69
    - tracing support, 72
    - transaction processing, 62
    - updating data, 46
    - using the Simple code sample, 76
    - using the Table Viewer code sample, 79
    - versions supported, 39
  - .NET database programming interfaces
    - tutorial, 83
  - 64-bit
    - ODBC, 368
  - @HttpMethod
    - accessing HTTP headers, 748
  - @HttpQueryString
    - accessing HTTP headers, 748
  - @HttpStatus
    - accessing HTTP headers, 748
  - @HttpURI
    - accessing HTTP headers, 748
  - @HttpVersion
    - accessing HTTP headers, 748
- ## A
- a\_backup\_db structure [database tools API]
    - description, 867

- a\_change\_log structure [database tools API]
    - description, 869
  - a\_create\_db structure [database tools API]
    - description, 871
  - a\_db\_info structure [database tools API]
    - description, 874
  - a\_db\_version\_info structure [database tools API]
    - description, 877
  - a\_dblic\_info structure [database tools API]
    - description, 877
  - a\_dbtools\_info structure [database tools API]
    - description, 878
  - a\_log\_file\_info structure [database tools API]
    - description, 879
  - a\_name structure [database tools API]
    - description, 880
  - a\_remote\_sql structure [database tools API]
    - description, 880
  - a\_sqlany\_bind\_param structure [SQL Anywhere C API]
    - description, 555
  - a\_sqlany\_bind\_param\_info structure [SQL Anywhere C API]
    - description, 555
  - a\_sqlany\_column\_info structure [SQL Anywhere C API]
    - description, 556
  - a\_sqlany\_data\_direction enumeration [SQL Anywhere C API]
    - description, 552
  - a\_sqlany\_data\_info structure [SQL Anywhere C API]
    - description, 557
  - a\_sqlany\_data\_type enumeration [SQL Anywhere C API]
    - description, 552
  - a\_sqlany\_data\_value structure [SQL Anywhere C API]
    - description, 558
  - a\_sqlany\_native\_type enumeration [SQL Anywhere C API]
    - description, 553
  - a\_sync\_db structure [database tools API]
    - description, 887
  - a\_syncpub structure [database tools API]
    - description, 896
  - a\_sysinfo structure [database tools API]
    - description, 897
  - a\_table\_info structure [database tools API]
    - description, 898
  - a\_translate\_log structure [database tools API]
    - description, 898
  - a\_truncate\_log structure [database tools API]
    - description, 902
  - a\_validate\_db structure [database tools API]
    - description, 903
  - Abort property
    - SARowsCopiedEventArgs class [SQL Anywhere .NET API], 299
  - Accept
    - accessing HTTP headers, 747
  - Accept-Charset
    - accessing HTTP headers, 747
  - Accept-Encoding
    - accessing HTTP headers, 747
  - Accept-Language
    - accessing HTTP headers, 747
  - AcceptCharset option
    - example, 759
- access-bridge.jar
  - deploying Java-based administration tools, 964
- accessibility
  - deploying Java-based administration tools, 957
- accessibility.properties
  - deploying Java-based administration tools, 964
- accessing Java methods
  - Java in the database, 391
- ActiveX Data Objects
  - about, 330
- Add method
  - SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 117
  - SAParameterCollection class [SQL Anywhere .NET API], 279
- addBatch
  - PreparedStatement class, 424
  - Statement class, 420
- adding
  - JAR files, 395
  - Java in the database classes, 394
- AddRange method
  - SAParameterCollection class [SQL Anywhere .NET API], 283
- addShutdownHook
  - Java VM shutdown hooks, 401
- AddWithValue method

---

- SAParameterCollection class [SQL Anywhere .NET API], 284
- administration tools
  - dbtools, 845
  - deploying, 956
- ADO
  - about, 330
  - Command object, 331
  - commands, 331
  - Connection object, 330
  - connections, 330
  - cursor types, 14
  - cursors, 31
  - introduction to programming, 329
  - queries, 332
  - Recordset object, 332
  - Recordset object and cursor types, 333
  - transactions, 335
  - updates, 334
  - updating data through a cursor, 334
  - using SQL statements in applications, 1
- ADO.NET
  - about, 39
  - autocommit mode, 35
  - controlling autocommit behavior, 35
  - cursor support, 31
  - deploying, 927
  - prepared statements, 3
  - using SQL statements in applications, 1
- ADO.NET API
  - about, 39
- alignment of data
  - ODBC, 372
- All property
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 158
- alloc\_sqllda function
  - about, 495
- alloc\_sqllda\_noind function
  - about, 496
- allowPasswordsInFavorites
  - configurable option, 975
- ALTER EXTERNAL ENVIRONMENT statement
  - using, 389
- altering
  - web services, 736
- an\_erase\_db structure [database tools API]
  - description, 905
- an\_extfn\_api
  - external function call interface, 568
- an\_extfn\_result\_set\_column\_data
  - external function call interface, 573
- an\_extfn\_result\_set\_column\_info
  - external function call interface, 572
- an\_extfn\_result\_set\_info
  - external function call interface, 571
- an\_extfn\_value
  - external function call interface, 570
- an\_unload\_db structure [database tools API]
  - description, 906
- an\_upgrade\_db structure [database tools API]
  - description, 911
- ANSI ODBC driver
  - Unix, 354
- Apache
  - choosing a PHP extension, 951
  - installing, 949
- apache\_files.txt
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- apache\_license\_1.1.txt
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- apache\_license\_2.0.txt
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- APIs
  - ADO API, 329
  - ADO.NET, 39
  - C API, 527
  - JDBC API, 403
  - ODBC API, 347
  - OLE DB API, 329
  - Perl DBD::SQLAnywhere API, 611
  - PHP, 640
  - Python Database API, 619
  - Ruby APIs, 691
  - Sybase Open Client API, 721
- AppInfo property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 186
- applications
  - deploying, 915

- deploying .NET clients, 927
- deploying client applications, 927
- deploying embedded SQL, 945
- deploying JDBC clients, 947
- deploying ODBC, 936
- deploying OLE DB, 930
- deploying Open Client, 956
- deploying PHP clients, 949
- SQL, 1
- ARRAY clause
  - using the FETCH statement, 483
- array fetches
  - ESQL, 483
- asensitive cursors
  - about, 24
  - delete example, 17
  - introduction, 17
  - update example, 19
- ASP.NET
  - adding the provider schema to the database, 92
  - providers, about, 91
  - registering the connection string, 93
  - registering the SQL Anywhere ASP.NET providers, 94
  - using the SQL Anywhere ASP.NET providers, 91
  - white paper available at Sybase.com, 92
- autocommit
  - controlling, 35
  - implementation, 36
  - JDBC, 418
  - ODBC, 357
  - setting for transactions, 35
- autoCommit option
  - configurable option, 975
- AUTOINCREMENT
  - finding most recent row inserted, 12
- autoRefetch
  - configurable option, 975
- AutoStart property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 186
- AutoStop property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 187
- B**
- background processing
  - callback functions, 494
- backups
  - DBTools example, 850
  - embedded SQL functions, 494
- batch inserts
  - JDBC, 424
- BatchSize property
  - SABulkCopy class [SQL Anywhere .NET API], 106
- BatchUpdateException
  - JDBC, 420
- batik-awt-util.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-bridge.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-codec.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-css.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-dom.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-ext.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-extension.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-gui-util.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-gvt.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- batik-parser.jar

- 
- deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
  - batik-script.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-svg-dom.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-svggen.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-swing.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-transcoder.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-util.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - batik-xml.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - BeginExecuteNonQuery method
    - SACommand class [SQL Anywhere .NET API], 126
  - BeginExecuteReader method
    - SACommand class [SQL Anywhere .NET API], 128
  - BeginTransaction method
    - SAConnection class [SQL Anywhere .NET API], 163
  - BIGINT data type
    - embedded SQL, 453
  - binary data types
    - embedded SQL, 453
  - bind parameters
    - prepared statements, 3
  - bind variables
    - about, 466
  - binding parameters
    - ODBC, 365
    - SQLBindParameter function, 367
  - BIT data type
    - embedded SQL, 453
  - bit fields
    - using, 849
  - blank padding
    - strings in embedded SQL, 449
  - BLOBs
    - embedded SQL, 487
    - retrieving in embedded SQL, 488
    - sending in embedded SQL, 490
  - block cursors
    - about, 11
    - ODBC, 15
  - bookmarks
    - about, 15
    - ODBC cursors, 378
  - bound columns
    - data alignment, 372
    - ODBC, 376
    - procedure result sets, 378
  - bound parameters
    - data alignment, 372
    - ODBC, 365
    - prepared statements, 3
    - procedure parameters, 378
    - SQLBindParameter function, 367
  - Broadcast property
    - SATcpOptionsBuilder class [SQL Anywhere .NET API], 308
  - BroadcastListener property
    - SATcpOptionsBuilder class [SQL Anywhere .NET API], 308
  - buffer length
    - ODBC, 368
  - Bulk-Library
    - about, 721
  - BulkCopyTimeout property
    - SABulkCopy class [SQL Anywhere .NET API], 106
  - byte code
    - Java classes, 386
- ## C
- C API
-

- introduction to programming, 527
- C programming language
  - data types, 453
  - embedded SQL applications, 435
- C#
  - support in .NET Data Provider, 39
- C++ applications
  - dbtools, 845
  - embedded SQL, 435
- C\_ESQL32 keyword
  - external environment, 588
- C\_ESQL64 keyword
  - external environment, 588
- C\_ODBC32 keyword
  - external environment, 588
- C\_ODBC64 keyword
  - external environment, 588
- CALL statement
  - embedded SQL, 491
- callback
  - JDBC, 427
- callback functions
  - embedded SQL, 494
  - registering, 510
- callbacks
  - DB\_CALLBACK\_CONN\_DROPPED, 511
  - DB\_CALLBACK\_DEBUG\_MESSAGE, 510
  - DB\_CALLBACK\_FINISH, 511
  - DB\_CALLBACK\_MESSAGE, 511
  - DB\_CALLBACK\_START, 510
  - DB\_CALLBACK\_VALIDATE\_FILE\_TRANSFERR, 512
  - DB\_CALLBACK\_WAIT, 511
- calling external libraries from procedures
  - about, 563
- calling Java methods
  - Java in the database, 391
- Cancel method
  - SACommand class [SQL Anywhere .NET API], 132
- cancel processing
  - in external functions, 576
- canceling requests
  - embedded SQL, 494
- CanCreateDataSourceEnumerator property
  - SAFactory class [SQL Anywhere .NET API], 254
- capabilities
  - supported, 727
- CD-ROM
  - deploying databases on, 984
- chained mode
  - controlling, 35
  - implementation, 36
  - transactions, 35
- chained option
  - JDBC, 418
- ChangeDatabase method
  - SAConnection class [SQL Anywhere .NET API], 165
- ChangePassword method
  - SAConnection class [SQL Anywhere .NET API], 166
- character data
  - character sets in Embedded SQL, 456
  - length in Embedded SQL, 456
- character sets
  - setting CHAR character set, 501
  - setting NCHAR character set, 502
  - web services, 759
- character strings
  - embedded SQL, 437
- Charset property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 187
- CharsetConversion option
  - example, 759
- checkForUpdates
  - configurable option, 975
- Checkpoint enumeration [database tools API]
  - description, 862
- cis.zip
  - deploying database servers, 979
- Class.forName method
  - loading SQL Anywhere JDBC 3.0 driver, 407
  - loading SQL Anywhere JDBC 4.0 driver, 408
- classes
  - creating, 394
  - installing, 393
  - updating, 396
  - versions, 396
- CLASSPATH environment variable
  - Java in the database, 391
- jConnect, 409
  - setting, 415
- clauses
  - WITH HOLD, 9



---

Clear method  
     SAParameterCollection class [SQL Anywhere .NET API], 284  
 ClearAllPools method  
     SAConnection class [SQL Anywhere .NET API], 166  
 clearBatch  
     Statement class, 420  
 ClearPool method  
     SAConnection class [SQL Anywhere .NET API], 166  
 client  
     time change, 519  
 client files  
     ESQL client API callback function, 510  
     ODBC client API callback function, 361  
 client side autocommit  
     about, 36  
 Client-Library  
     Sybase Open Client, 721  
 CLIENTPORT clause  
     specifying, 774  
 ClientPort property  
     SATcpOptionsBuilder class [SQL Anywhere .NET API], 309  
 clients  
     web, 766  
 Close method  
     SABulkCopy class [SQL Anywhere .NET API], 103  
     SAConnection class [SQL Anywhere .NET API], 167  
     SADataReader class [SQL Anywhere .NET API], 217  
 close method  
     Python, 621  
 CLOSE statement  
     using cursors in embedded SQL, 480  
 CLR  
     external environment, 584  
 CLR keyword  
     external environment, 584  
 CodeXchange  
     samples, 730  
 colon host variable name  
     SQLBindParameter function, 367  
 ColumnMappings property  
     SABulkCopy class [SQL Anywhere .NET API], 106  
 Columns field  
     SAMetaDataCollectionNames class [SQL Anywhere .NET API], 259  
 Command ADO object  
     ADO, 331  
 command line utilities  
     deploying, 989  
     SQL preprocessor (sqlpp) syntax, 437  
 Command property  
     SARowUpdatedEventArgs class [SQL Anywhere .NET API], 301  
     SARowUpdatingEventArgs class [SQL Anywhere .NET API], 303  
 commands  
     ADO Command object, 331  
 CommandText property  
     SACommand class [SQL Anywhere .NET API], 141  
 CommandTimeout property  
     SACommand class [SQL Anywhere .NET API], 141  
 CommandType property  
     SACommand class [SQL Anywhere .NET API], 141  
 CommBufferSize property  
     SAConnectionStringBuilder class [SQL Anywhere .NET API], 187  
 commenting  
     web services, 740  
 Commit method  
     SATransaction class [SQL Anywhere .NET API], 313  
 commit method  
     Python, 623  
 COMMIT statement  
     cursors, 37  
     JDBC, 418  
 commitOnExit  
     configurable option, 975  
 committing  
     transactions from ODBC, 357  
 CommitTrans ADO method  
     ADO programming, 335  
     updating data, 335  
 CommLinks property

- SAConnectionStringBuilder class [SQL Anywhere .NET API], 187
- compile and link process
  - about, 436
- compilers
  - used with sqlpp, 441
- Compress property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 188
- CompressionThreshold property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 188
- concurrency values
  - SQL\_CONCUR\_LOCK, 376
  - SQL\_CONCUR\_READ\_ONLY, 376
  - SQL\_CONCUR\_ROWVER, 376
  - SQL\_CONCUR\_VALUES, 376
- configuring
  - administration tools for deployment, 975
  - Interactive SQL for deployment, 975
  - Sybase Central for deployment, 975
- conformance
  - ODBC, 348
- connect method
  - Python, 621
- Connection
  - accessing HTTP headers, 747
- Connection ADO object
  - ADO, 330
  - ADO programming, 335
- connection handles
  - ODBC, 355
- connection parameters
  - OLE DB, 336
- connection pooling
  - .NET Data Provider, 45
  - OLE DB, 338
  - web services, 743
- connection profiles
  - deploying administration tools on Linux, Solaris, and Mac OS X, 974
  - deploying administration tools on Windows, 966
- connection properties
  - web services, 761
- Connection property
  - SACommand class [SQL Anywhere .NET API], 142
  - SATransaction class [SQL Anywhere .NET API], 315
- connection state
  - .NET Data Provider, 46
- CONNECTION\_PROPERTY function
  - example, 757
- ConnectionLifetime property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 188
- ConnectionName property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 188
- ConnectionPool property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 189
- ConnectionReset property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 189
- connections
  - ADO Connection object, 330
  - connecting to a database using the .NET Data Provider, 43
  - functions, 516
  - jConnect, 411
  - jConnect URL, 410
  - JDBC, 406
  - JDBC client applications, 412
  - JDBC defaults, 418
  - JDBC example, 412
  - JDBC in the server, 415
  - JDBC server-side example, 415
  - JDBC transaction isolation level, 418
  - licensing web applications, 752
  - ODBC functions, 357
  - ODBC getting attributes, 360
  - ODBC programming, 358
  - ODBC setting attributes, 360
  - SQL Anywhere JDBC driver URL, 408
- ConnectionString property
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 159
  - SAConnection class [SQL Anywhere .NET API], 176
- ConnectionTimeout property
  - SAConnection class [SQL Anywhere .NET API], 177
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 189

---

console utility (dbconsole)  
 deploying, 956  
 deploying on Linux and Unix, 967  
 deploying on Windows, 957

Contains method  
 SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 120  
 SAParameterCollection class [SQL Anywhere .NET API], 284

ContainsKey method  
 SAConnectionStringBuilderBase class [SQL Anywhere .NET API], 200

conventions  
 file names, 918

conversion  
 data types, 460

cookies  
 creating, 753  
 session management, 755

CopyTo method  
 SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 120  
 SAErrorCollection class [SQL Anywhere .NET API], 245  
 SAParameterCollection class [SQL Anywhere .NET API], 285

corrupt databases  
 EnableFlush registry entry, 981

Count property  
 SAErrorCollection class [SQL Anywhere .NET API], 245  
 SAParameterCollection class [SQL Anywhere .NET API], 289

create database wizard  
 deployment considerations, 957

create Java class wizard  
 using, 394

CREATE PROCEDURE statement  
 embedded SQL, 491

CreateCommand method  
 SAConnection class [SQL Anywhere .NET API], 167  
 SAFactory class [SQL Anywhere .NET API], 251

CreateCommandBuilder method  
 SAFactory class [SQL Anywhere .NET API], 251

CreateConnection method  
 SAFactory class [SQL Anywhere .NET API], 252

CreateConnectionStringBuilder method  
 SAFactory class [SQL Anywhere .NET API], 252  
 CreateDataAdapter method  
 SAFactory class [SQL Anywhere .NET API], 253

CreateDataSourceEnumerator method  
 SAFactory class [SQL Anywhere .NET API], 253

CreateParameter method  
 SACommand class [SQL Anywhere .NET API], 132  
 SAFactory class [SQL Anywhere .NET API], 253  
 using, 3

CreatePermission method  
 SAFactory class [SQL Anywhere .NET API], 254  
 SAPermissionAttribute class [SQL Anywhere .NET API], 297

creating  
 procedures and functions with external calls, 563  
 web services, 736

CS-Library  
 about, 721

CS\_CSR\_ABS  
 not supported with Open Client, 727

CS\_CSR\_FIRST  
 not supported with Open Client, 727

CS\_CSR\_LAST  
 not supported with Open Client, 727

CS\_CSR\_PREV  
 not supported with Open Client, 727

CS\_CSR\_REL  
 not supported with Open Client, 727

CS\_DATA\_BOUNDARY  
 not supported with Open Client, 727

CS\_DATA\_SENSITIVITY  
 not supported with Open Client, 727

CS\_PROTO\_DYNPROC  
 not supported with Open Client, 727

CS\_REG\_NOTIF  
 not supported with Open Client, 727

CS\_REQ\_BCP  
 not supported with Open Client, 727

ct\_command function  
 describing results in Open Client, 726  
 executing statements in Open Client, 725

ct\_cursor function  
 Open Client, 725

ct\_dynamic function  
 Open Client, 725

ct\_results function  
 Open Client, 726

- ct\_send function
  - Open Client, 726
- cursor positioning
  - troubleshooting, 9
- cursor sensitivity and performance
  - about, 26
- cursors
  - about, 6
  - ADO, 31
  - ADO.NET, 31
  - asensitive, 24
  - availability, 14
  - benefits, 7
  - block cursors, 15
  - canceling, 14
  - choosing ODBC cursor characteristics, 375
  - db\_cancel\_request function, 501
  - delete, 726
  - describing result sets, 33
  - determining what cursors exist for a connection, 5
  - dynamic, 22
  - DYNAMIC SCROLL and asensitive cursors, 24
  - DYNAMIC SCROLL and cursor positioning, 9
  - embedded SQL supported types, 32
  - embedded SQL usage, 480
  - example C code, 445
  - fat, 11
  - fetching multiple rows, 11
  - fetching rows, 10
  - insensitive, 21
  - inserting multiple rows, 12
  - inserting rows, 11
  - internals, 16
  - isolation level, 9
  - keyset-driven, 24
  - limitations, 9
  - membership, 16
  - ODBC, 374
  - ODBC and SQL Anywhere types, 31
  - ODBC bookmarks, 378
  - ODBC deletes, 377
  - ODBC result sets, 376
  - ODBC updates, 377
  - OLE DB, 31
  - Open Client, 725
  - order, 16
  - platforms, 14
  - positioning, 9
  - prefetch performance, 27
  - prepared statements, 7
  - properties, 15
  - Python, 622
  - read-only, 21
  - requesting, 31
  - result sets, 5
  - savepoints, 38
  - SCROLL, 24
  - scrollability, 15
  - scrollable, 11
  - sensitive, 22
  - sensitivity, 15
  - sensitivity and deletion example, 17
  - sensitivity and isolation levels, 30
  - sensitivity and performance, 26
  - sensitivity and update example, 19
  - sensitivity in SQL Anywhere, 16
  - sensitivity overview, 17
  - static, 21
  - stored procedures, 492
  - transactions, 37
  - uniqueness, 15
  - unspecified sensitivity, 24
  - updatability, 15
  - update, 726
  - updating, 334
  - updating and deleting rows, 11
  - uses, 5
  - using, 7
  - value-sensitive, 24
  - values, 16
  - viewing contents of cursors for a connection, 5
  - visible changes, 16
  - work tables, 26
- cursors and bookmarks
  - about, 15
- D**
- data
  - accessing with the .NET Data Provider, 46
  - manipulating with the .NET Data Provider, 46
- data connection
  - Visual Studio, 84
- data sources
  - deploying, 942
  - file DSN, 942

---

- system DSN, 942
- user DSN, 942
- data type conversions
  - indicator variables, 460
- data types
  - C data types, 453
  - dynamic SQL, 469
  - embedded SQL, 449
  - host variables, 453
  - in web services handlers, 789
  - mapping, 723
  - Open Client, 723
  - ranges, 723
  - SQL and C, 578
  - SQLDA, 471
- DataAdapter
  - about, 46
  - deleting data, 52
  - inserting data, 52
  - obtaining primary key values, 57
  - retrieving data, 51
  - updating data, 52
  - using, 51
- DataAdapter property
  - SACommandBuilder class [SQL Anywhere .NET API], 154
- database management
  - dbtools, 845
- database options
  - set for jConnect, 412
- database properties
  - db\_get\_property function, 505
- Database property
  - SAConnection class [SQL Anywhere .NET API], 178
- database servers
  - deploying, 979
  - functions, 516
- database tools API
  - a\_backup\_db structure, 867
  - a\_change\_log structure, 869
  - a\_create\_db structure, 871
  - a\_db\_info structure, 874
  - a\_db\_version\_info structure, 877
  - a\_dblic\_info structure, 877
  - a\_dbtools\_info structure, 878
  - a\_log\_file\_info structure, 879
  - a\_name structure, 880
  - a\_remote\_sql structure, 880
  - a\_sync\_db structure, 887
  - a\_syncpub structure, 896
  - a\_sysinfo structure, 897
  - a\_table\_info structure, 898
  - a\_translate\_log structure, 898
  - a\_truncate\_log structure, 902
  - a\_validate\_db structure, 903
  - an\_erase\_db structure, 905
  - an\_unload\_db structure, 906
  - an\_upgrade\_db structure, 911
  - Checkpoint enumeration, 862
  - DBBackup method, 853
  - DBChangeLogName method, 853
  - DBCreate method, 854
  - DBCreatedVersion method, 854
  - DBErase method, 855
  - DBInfo method, 855
  - DBInfoDump method, 856
  - DBInfoFree method, 856
  - DBLicense method, 857
  - DBLogFileInfo method, 857
  - DBRemoteSQL method, 858
  - DBSynchronizeLog method, 858
  - DBToolsFini method, 858
  - DBToolsInit method, 859
  - DBToolsVersion method, 860
  - DBTranslateLog method, 860
  - DBTruncateLog method, 860
  - DBUnload method, 861
  - DBUpgrade method, 861
  - DBValidate method, 862
  - Padding enumeration, 863
  - Unit enumeration, 864
  - Unload enumeration, 864
  - UserList enumeration, 865
  - Validation enumeration, 865
  - Verbosity enumeration, 866
  - Version enumeration, 867
- database tools interface
  - about, 845
- database tools library
  - about, 845
- DatabaseFile property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 189
- DatabaseKey property

- SAConnectionStringBuilder class [SQL Anywhere .NET API], 190
- DatabaseName property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 190
- databases
  - deploying, 983
  - installing jConnect metadata support, 409
  - storing Java classes, 386
  - URL, 411
- DatabaseSwitches property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 190
- datagrid control
  - Visual Studio, 88
- DataSet
  - SQL Anywhere .NET Data Provider, 52
- DataSource property
  - SAConnection class [SQL Anywhere .NET API], 178
- DataSourceInformation field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 260
- DataSourceName property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 190
- DataTypes field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 260
- DATETIME data type
  - embedded SQL, 453
- DB-Library
  - about, 721
- DB\_ACTIVE\_CONNECTION
  - db\_find\_engine function, 504
- db\_backup function
  - about, 496
  - dbbackup utility, 494
- DB\_BACKUP\_CLOSE\_FILE parameter
  - about, 496
- DB\_BACKUP\_END parameter
  - about, 496
- DB\_BACKUP\_INFO parameter
  - about, 496
- DB\_BACKUP\_INFO\_CHKPT\_LOG parameter
  - about, 496
- DB\_BACKUP\_INFO\_PAGES\_IN\_BLOCK parameter
  - about, 496
- DB\_BACKUP\_OPEN\_FILE parameter
  - about, 496
- DB\_BACKUP\_PARALLEL\_READ parameter
  - about, 496
- DB\_BACKUP\_PARALLEL\_START parameter
  - about, 496
- DB\_BACKUP\_READ\_PAGE parameter
  - about, 496
- DB\_BACKUP\_READ\_RENAME\_LOG parameter
  - about, 496
- DB\_BACKUP\_START parameter
  - about, 496
- DB\_CALLBACK\_CONN\_DROPPED callback parameter
  - about, 511
- DB\_CALLBACK\_DEBUG\_MESSAGE callback parameter
  - about, 510
- DB\_CALLBACK\_FINISH callback parameter
  - about, 511
- DB\_CALLBACK\_MESSAGE callback parameter
  - about, 511
- DB\_CALLBACK\_START callback parameter
  - about, 510
- DB\_CALLBACK\_VALIDATE\_FILE\_TRANSFER callback parameter
  - about, 512
- DB\_CALLBACK\_WAIT callback parameter
  - about, 511
- DB\_CAN\_MULTI\_CONNECT
  - db\_find\_engine function, 504
- DB\_CAN\_MULTI\_DB\_NAME
  - db\_find\_engine function, 504
- db\_cancel\_request function
  - about, 501
  - request management, 494
- db\_change\_char\_charset function
  - about, 501
- db\_change\_nchar\_charset function
  - about, 502
- DB\_CLIENT
  - db\_find\_engine function, 504
- DB\_CONNECTION\_DIRTY
  - db\_find\_engine function, 504
- DB\_DATABASE\_SPECIFIED
  - db\_find\_engine function, 504
- db\_delete\_file function

---

- about, 503
- DB\_ENGINE
  - db\_find\_engine function, 504
- db\_find\_engine function
  - about, 503
- db\_fini function
  - about, 504
- db\_fini\_dll
  - calling, 444
- db\_get\_property function
  - about, 505
- db\_init function
  - about, 506
- db\_init\_dll
  - calling, 444
- db\_is\_working function
  - about, 507
  - request management, 494
- db\_locate\_servers function
  - about, 507
- db\_locate\_servers\_ex function
  - about, 508
- DB\_LOOKUP\_FLAG\_ADDRESS\_INCLUDES\_PO  
RT
  - about, 508
- DB\_LOOKUP\_FLAG\_DATABASES
  - about, 508
- DB\_LOOKUP\_FLAG\_NUMERIC
  - about, 508
- DB\_NO\_DATABASES
  - db\_find\_engine function, 504
- DB\_PROP\_CLIENT\_CHARSET
  - usage, 505
- DB\_PROP\_DBLIB\_VERSION
  - usage, 505
- DB\_PROP\_SERVER\_ADDRESS
  - usage, 505
- db\_register\_a\_callback function
  - about, 510
  - request management, 494
- db\_start\_database function
  - about, 513
- db\_start\_engine function
  - about, 514
- db\_stop\_database function
  - about, 515
- db\_stop\_engine function
  - about, 516
- db\_string\_connect function
  - about, 516
- db\_string\_disconnect function
  - about, 517
- db\_string\_ping\_server function
  - about, 518
- db\_time\_change function
  - about, 519
- DBBackup method [database tools API]
  - description, 853
- dbcapi.dll
  - deploying, 985
  - deploying PHP clients, 949
  - PHP support module, 607
- DBChangeLogName method [database tools API]
  - description, 853
- dbcis12.dll
  - deploying database servers, 979
- dbclrenv12.dll
  - deploying, 986
- dbcon12.dll
  - deploying database utilities, 989
  - deploying embedded SQL clients, 946
  - deploying ODBC clients, 937
  - deploying OLE DB clients, 930
  - deploying on Windows, 960
  - deploying PHP clients, 949
- dbconsole utility
  - deploying, 956
  - deploying administration tools on Linux/Unix/Mac OS X, 972
  - deploying administration tools on Windows, 963
  - deploying on Linux and Unix, 967
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 957
- dbconsole.exe
  - deploying on Windows, 960
- dbconsole.ini
  - deploying administration tools, 956
- DBConsole.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- DBCreate method [database tools API]
  - description, 854
- DBCreatedVersion method [database tools API]
  - description, 854

- dbctrs12.dll
  - deploying database servers, 979
  - deploying SQL Anywhere, 985
- DBD::SQLAnywhere
  - about, 611
  - connecting to a database, 615
  - executing SQL statements, 616
  - handling multiple result sets, 617
  - inserting rows, 618
  - installing on Unix and Mac OS X, 613
  - installing on Windows, 611
  - writing Perl scripts, 614
- dbdata.dll
  - SQL Anywhere .NET Data Provider, 71
- dbecc12.dll
  - deploying embedded SQL clients, 946
  - deploying PHP clients, 949
  - ECC encryption, 987
- dbelevate12.exe
  - deploying on Windows, 960
  - registering DLLs, 984
- dbencod12.dll
  - deploying SQL Remote, 991
- dbeng12
  - deploying database servers, 979
- dbeng12.exe
  - deploying database servers, 979
- dbeng12.lic
  - deploying database servers, 979
- DBErase method [database tools API]
  - description, 855
- dbextclr12.exe
  - deploying, 986
- dbextenv12.dll
  - deploying, 985
  - PHP support module, 607
- dbexternc12
  - deploying, 985
  - external calls, 589
- dbextf.dll
  - deploying database servers, 979
- dbfile12.dll
  - deploying SQL Remote, 991
- dbfips12.dll
  - deploying embedded SQL clients, 946
  - deploying PHP clients, 949
  - FIPS-certified AES encryption, 987
  - FIPS-certified RSA encryption, 987
- dbftp12.dll
  - deploying SQL Remote, 991
- dbghelp.dll
  - deploying database servers, 979
  - deploying on Windows, 960
- dbicu12.dll
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Windows, 960
- dbicudt12.dat
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
- dbicudt12.dll
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Windows, 960
- DBInfo method [database tools API]
  - description, 855
- DBInfoDump method [database tools API]
  - description, 856
- DBInfoFree method [database tools API]
  - description, 856
- dbinit utility
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deployment considerations, 989
- dbinit.exe
  - deploying on Windows, 960
- dbisql utility
  - deploying administration tools on Windows, 960
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
- dbisql.com
  - deploying on Windows, 960
- dbisql.exe
  - deploying on Windows, 960
- dbisql.ini
  - deploying administration tools, 956
- dbisqlc utility
  - deploying, 977
  - limited functionality, 977
  - Unix supported deployment platforms, 967
- dbjdbc12.dll
  - deploying, 985
  - deploying database servers, 979



---

- deploying JDBC clients, 948
- dbjodbc12.dll
  - deploying database servers, 979
  - deploying on Windows, 960
- dblgen12.dll
  - deploying database servers, 979
  - deploying database utilities, 989
  - deploying embedded SQL clients, 946
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying OLE DB clients, 930
  - deploying on Windows, 960
  - deploying PHP clients, 949
  - deploying SQL Remote, 991
- dblgen12.dll registry entry
  - about, 981
- dblgen12.res
  - deploying database servers, 979
  - deploying database utilities, 989
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying SQL Remote, 991
- DBLIB
  - dynamic loading, 444
  - interface library, 435
- dblib12.dll
  - deploying database utilities, 989
  - deploying embedded SQL clients, 946
  - deploying on Windows, 960
  - deploying PHP clients, 949
  - deploying SQL Remote, 991
- DBLicense method [database tools API]
  - description, 857
- DBLogFileInfo method [database tools API]
  - description, 857
- dbodbc12.bundle
  - deploying ODBC clients, 937
  - Mac OS X ODBC driver, 352
- dbodbc12.dll
  - deploying database servers, 979
  - deploying ODBC clients, 937
  - deploying on Windows, 960
  - deploying SQL Anywhere, 985
  - linking, 349
- dbodbc12.lib
  - Windows Mobile ODBC import library, 350
- dbodbc12\_r.bundle
  - deploying ODBC clients, 937
- dboftsp.dll
  - deploying database utilities, 989
- dboledb12.dll
  - deploying OLE DB clients, 930
  - deploying SQL Anywhere, 985
- dboledba12.dll
  - deploying OLE DB clients, 930
  - deploying SQL Anywhere, 985
- DbProviderFactory
  - registering, 71
- dbput12.dll
  - deploying on Windows, 960
- dbremote utility
  - deploying SQL Remote, 991
- DBRemoteSQL method [database tools API]
  - description, 858
- dbrmt.h
  - about, 845
- dbrsa12.dll
  - RSA encryption, 987
- dbrsakp12.dll
  - deploying database servers, 979
  - deploying JDBC clients, 947
  - deploying Open Client, 956
- dbscript12.dll
  - deploying database servers, 979
- dbserv12.dll
  - AES encryption, 987
  - deploying database servers, 979
- dbsmtp12.dll
  - deploying SQL Remote, 991
- dbsrv12
  - deploying database servers, 979
- dbsrv12.exe
  - deploying database servers, 979
- dbsrv12.lic
  - deploying database servers, 979
- DBSynchronizeLog method [database tools API]
  - description, 858
- dbtool12.dll
  - about, 845
  - deploying database utilities, 989
  - deploying on Windows, 960
  - deploying SQL Remote, 991
  - Windows Mobile, 845
- DBTools interface

- about, 845
- calling DBTools functions, 847
- example program, 850
- finalizing, 846
- finishing, 846
- initializing, 846
- introduction, 845
- return codes, 912
- starting, 846
- using, 845
- dbtools.h
  - about, 845
- dbtools.h header file
  - SQL Anywhere Database Tools C API reference, 852
- DBToolsFini method [database tools API]
  - description, 858
- DBToolsInit method [database tools API]
  - description, 859
- DBToolsVersion method [database tools API]
  - description, 860
- DBTranslateLog method [database tools API]
  - description, 860
- DBTruncateLog method [database tools API]
  - description, 860
- DbType property
  - SAPParameter class [SQL Anywhere .NET API], 272
- DBUnload method [database tools API]
  - description, 861
- dbunload utility
  - deployment considerations, 989
  - deployment for version 9 and earlier databases, 990
- dbunlspt
  - deployment for version 9 and earlier databases, 990
- dbupgrad utility
  - installing jConnect metadata support, 409
- DBUpgrade method [database tools API]
  - description, 861
- dbusde.dll
  - deployment for version 9 and earlier databases, 990
- dbusde.res
  - deployment for version 9 and earlier databases, 990
- dbusen.dll
  - deployment for version 9 and earlier databases, 990
- dbusen.res
  - deployment for version 9 and earlier databases, 990
- dbuses.dll
  - deployment for version 9 and earlier databases, 990
- deployment for version 9 and earlier databases, 990
- dbusfr.dll
  - deployment for version 9 and earlier databases, 990
- dbusfr.res
  - deployment for version 9 and earlier databases, 990
- dbusit.dll
  - deployment for version 9 and earlier databases, 990
- dbusja.dll
  - deployment for version 9 and earlier databases, 990
- dbusja.res
  - deployment for version 9 and earlier databases, 990
- dbusko.dll
  - deployment for version 9 and earlier databases, 990
- dbuslt.dll
  - deployment for version 9 and earlier databases, 990
- dbuspl.dll
  - deployment for version 9 and earlier databases, 990
- dbuspt.dll
  - deployment for version 9 and earlier databases, 990
- dbusru.dll
  - deployment for version 9 and earlier databases, 990
- dbustw.dll
  - deployment for version 9 and earlier databases, 990
- dbusuk.dll
  - deployment for version 9 and earlier databases, 990
- dbuszh.res
  - deployment for version 9 and earlier databases, 990
- DBValidate method [database tools API]
  - description, 862
- debugger.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- DECIMAL data type
  - embedded SQL, 453
- DECL\_BIGINT macro
  - about, 453
- DECL\_BINARY macro
  - about, 453
- DECL\_BIT macro
  - about, 453
- DECL\_DATETIME macro
  - about, 453
- DECL\_DECIMAL macro
  - about, 453
- DECL\_FIXCHAR macro
  - about, 453
- DECL\_LONGBINARY macro

---

- about, 453
- DECL\_LONGNVARCHAR macro
  - about, 453
- DECL\_LONGVARCHAR macro
  - about, 453
- DECL\_NCHAR macro
  - about, 453
- DECL\_NFIXCHAR macro
  - about, 453
- DECL\_NVARCHAR macro
  - about, 453
- DECL\_UNSIGNED\_BIGINT macro
  - about, 453
- DECL\_VARCHAR macro
  - about, 453
- declaration section
  - about, 452
- DECLARE section
  - about, 452
- DECLARE statement
  - using cursors in embedded SQL, 480
- declaring
  - embedded SQL data types, 449
  - host variables, 452
- DefaultDSNDir
  - ODBC FileDSN, 942
- DELETE statement
  - JDBC, 420
  - positioned, 11
- DeleteCommand property
  - SADDataAdapter class [SQL Anywhere .NET API], 210
- DeleteDynamic method
  - JDBCExample, 423
- DeleteStatic method
  - JDBCExample, 421
- deploying
  - .NET data provider, 927
  - about, 915
  - administration tools, 956
  - administration tools on Linux and Unix, 967
  - administration tools on Windows, 957
  - ADO.NET data provider, 927
  - applications and databases, 915
  - client applications, 927
  - CLR external environment, 986
  - console utility (dbconsole), 956
  - console utility (dbconsole) on Linux and Unix, 967
  - console utility (dbconsole) on Windows, 957
  - database servers, 979
  - databases, 983
  - databases on CD-ROM, 984
  - deploying, 985
  - deployment wizard, 919
  - embedded databases, 988
  - embedded SQL, 945
  - ESQL external environment, 985
  - external environment, 985
  - file locations, 916
  - Interactive SQL, 956
  - Interactive SQL (dbisqlc), 977
  - Interactive SQL on Linux and Unix, 967
  - Interactive SQL on Windows, 957
  - international considerations, 983
  - Java external environment, 985
  - Java in the database, 979
  - jConnect, 947
  - JDBC clients, 947
  - localizing databases, 983
  - MobiLink plug-in, 957
  - ODBC, 936
  - ODBC data sources, 942
  - ODBC external environment, 985
  - OLE DB provider, 930
  - Open Client, 956
  - overview, 915
  - Perl external environment, 986
  - personal database server, 988
  - PHP clients, 949
  - PHP extension, 949
  - PHP external environment, 986
  - QAnywhere plug-in, 957
  - read-only databases, 984
  - registering DLLs, 984
  - registry settings, 981
  - silent install, 923
  - SQL Anywhere, 915
  - SQL Anywhere .NET Data Provider applications, 69
  - SQL Anywhere components on Windows, 919
  - SQL Anywhere JDBC driver, 947
  - SQL Anywhere plug-in, 957
  - SQL Remote, 991
  - Sybase Central, 956
  - Sybase Central on Linux and Unix, 967
  - Sybase Central on Windows, 957

- UltraLite plug-in, 957
- Unix issues, 916
- wizard, 919
- deploying on Linux and Unix
  - SQL Anywhere Console utility (dbconsole), 967
- deploying the SQL Anywhere .NET Data Provider
  - about, 69
- deployment wizard
  - about, 919
  - running, 921
- Depth property
  - SADataReader class [SQL Anywhere .NET API], 237
- DeriveParameters method
  - SACommandBuilder class [SQL Anywhere .NET API], 148
- DESCRIBE SELECT LIST statement
  - dynamic SELECT statement, 468
- DESCRIBE statement
  - DT\_HAS\_USERTYPE\_INFO, 472
  - multiple result sets, 493
  - SQLDA fields, 471
  - sqlen field, 473
  - sqltype field, 473
  - used in dynamic SELECT statements, 468
- describing
  - NCHAR columns in Embedded SQL, 473
  - result sets, 33
- descriptors
  - describing result sets, 33
- DesignTimeVisible property
  - SACommand class [SQL Anywhere .NET API], 142
- DestinationColumn property
  - SABulkCopyColumnMapping class [SQL Anywhere .NET API], 112
- DestinationOrdinal property
  - SABulkCopyColumnMapping class [SQL Anywhere .NET API], 113
- DestinationTableName property
  - SABulkCopy class [SQL Anywhere .NET API], 107
- developing applications with the .NET Data Provider
  - about, 39
- developing applications with the ASP.NET providers
  - about, 91
- Direction property
  - SAPParameter class [SQL Anywhere .NET API], 272
- directory structure
  - Unix, 916
- disableExecuteAll
  - configurable option, 975
- DisableMultiRowFetch property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 191
- disableResultsEditing
  - configurable option, 975
- DISH services
  - .NET tutorial, 826
  - about, 734
  - commenting, 740
  - creating, 738
  - dropping, 739
  - homogeneous, 739
  - JAX-WS tutorial, 818
  - SQL Anywhere web client tutorial, 810
- Dispose method
  - SABulkCopy class [SQL Anywhere .NET API], 103
- Distributed Transaction Coordinator
  - three-tier computing, 841
- distributed transaction processing
  - using the SQL Anywhere .NET Data Provider, 62
- distributed transactions
  - about, 839
  - architecture, 842
  - enlistment, 841
  - recovery, 844
  - restrictions, 843
  - three-tier computing, 840
- DLL entry points
  - about, 495
- DllMain
  - calling db\_fini, 504
  - calling ODBC functions, 363
- DLLs
  - calling from stored procedures, 563
  - deployment, 984
  - external procedure calls, 564
  - multiple SQLCAs, 465
  - registering for deployment, 984
- DoBroadcast property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 309

---

documentation

- deploying on Linux, 978
- deploying on Mac OS X, 978
- deploying on Unix, 978
- deploying on Windows, 978

Driver registry entry

- ODBC, 942

drivers

- deploying the SQL Anywhere ODBC driver, 936
- jConnect JDBC driver, 404
- linking the SQL Anywhere ODBC driver on Windows, 349
- SQL Anywhere JDBC driver, 404

dropping

- web services, 739

DSN-less connections

- using ODBC, 945

DT\_BIGINT embedded SQL data type

- about, 449

DT\_BINARY embedded SQL data type

- about, 451

DT\_BIT embedded SQL data type

- about, 449

DT\_DATE embedded SQL data type

- about, 449

DT\_DECIMAL embedded SQL data type

- about, 449

DT\_DOUBLE embedded SQL data type

- about, 449

DT\_FIXCHAR embedded SQL data type

- about, 449

DT\_FLOAT embedded SQL data type

- about, 449

DT\_HAS\_USERTYPE\_INFO

- DESCRIBE statement, 472

DT\_INT embedded SQL data type

- about, 449

DT\_LONGBINARY embedded SQL data type

- about, 451

DT\_LONGNVARCHAR embedded SQL data type

- about, 450

DT\_LONGVARCHAR embedded SQL data type

- about, 450

DT\_NFIXCHAR embedded SQL data type

- about, 450

DT\_NSTRING embedded SQL data type

- about, 449

DT\_NVARCHAR embedded SQL data type

- about, 450

DT\_PROCEDURE\_IN

- using, 493

DT\_PROCEDURE\_OUT

- using, 493

DT\_SMALLINT embedded SQL data type

- about, 449

DT\_STRING embedded SQL data type

- about, 449

DT\_TIME embedded SQL data type

- about, 449

DT\_TIMESTAMP embedded SQL data type

- about, 449

DT\_TIMESTAMP\_STRUCT embedded SQL data type

- about, 451

DT\_TINYINT embedded SQL data type

- about, 449

DT\_UNSBIGINT embedded SQL data type

- about, 449

DT\_UNSINT embedded SQL data type

- about, 449

DT\_UNSSMALLINT embedded SQL data type

- about, 449

DT\_VARCHAR embedded SQL data type

- about, 450

DT\_VARIABLE embedded SQL data type

- about, 451

DTC

- isolation levels, 843
- three-tier computing, 841

DTC isolation levels

- about, 843

dynamic cursors

- about, 22
- ODBC, 31
- sample, 447

DYNAMIC SCROLL cursors

- asensitive cursors, 24
- embedded SQL, 32
- troubleshooting, 9

dynamic SELECT statement

- DESCRIBE SELECT LIST statement, 468

dynamic SQL

- about, 466
- SQLDA, 469

**E**

- EAServer
  - three-tier computing, 841
- Elevate property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 191
- embedded databases
  - deploying, 988
- embedded SQL
  - about, 435
  - authorization, 437
  - autocommit mode, 35
  - calling db\_fini from DllMain, 504
  - character strings, 437
  - compile and link process, 436
  - controlling autocommit behavior, 35
  - cursor examples, 445
  - cursor types, 14
  - cursors, 32
  - deploying clients, 945
  - development, 435
  - dynamic cursors, 447
  - dynamic statements, 466
  - example program, 443
  - external environment, 587
  - FETCH FOR UPDATE, 30
  - fetching data, 479
  - functions, 495
  - header files, 441
  - host variables, 452
  - import libraries, 442
  - line numbers, 437
  - SQL statements, 1
  - statement summary, 525
  - static statements, 466
  - using cursors, 480
- EnableFlush registry entry
  - preventing database corruption, 981
- EncryptedPassword property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 191
- Encryption property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 191
- EndExecuteNonQuery method
  - SACCommand class [SQL Anywhere .NET API], 133
- EndExecuteReader method
  - SACCommand class [SQL Anywhere .NET API], 135
- Enlist property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 192
- EnlistDistributedTransaction method
  - SACConnection class [SQL Anywhere .NET API], 168
- enlistment
  - distributed transactions, 841
- EnlistTransaction method
  - SACConnection class [SQL Anywhere .NET API], 168
- Entity Framework
  - using, 64
- Entity Framework support
  - iAnywhere.Data.SQLAnywhere provider, 40
- entry points
  - calling DBTools functions, 847
- environment handles
  - ODBC, 355
- environment variables
  - for Java in the database, 389
- error codes
  - SQL Anywhere exit codes, 912
- error handling
  - Java, 388
  - ODBC, 382
  - SQL Anywhere .NET Data Provider, 63
- error messages
  - embedded SQL function, 524
- errors
  - HTTP codes, 804
  - SOAP faults, 804
  - sqlcode SQLCA field, 461
- Errors property
  - SAException class [SQL Anywhere .NET API], 248
  - SAInfoMessageEventArgs class [SQL Anywhere .NET API], 256
- escape syntax
  - Interactive SQL, 430
  - ODBC, 380
- esqldll.c
  - about, 444
- event log
  - EventLogMask, 982

---

- registry entry, 981
- exceptions
  - Java, 388
- EXEC SQL
  - embedded SQL development, 443
- execute method
  - Python, 622
- EXECUTE statement
  - stored procedures in embedded SQL, 491
  - using, 466
- executeBatch
  - PreparedStatement class, 424
  - Statement class, 420
- executemany method
  - Python, 623
- ExecuteNonQuery method
  - SACommand class [SQL Anywhere .NET API], 137
  - using SACommand, 48
- ExecuteReader method
  - ADO.NET prepared statements, 3
  - SACommand class [SQL Anywhere .NET API], 137
  - SACommand class Simple sample, 78
  - SACommand class TableView sample, 83
  - using SACommand, 47
- ExecuteScalar method
  - SACommand class [SQL Anywhere .NET API], 139
  - using SACommand, 47
- executeToolBarButtonSemantics
  - configurable option, 975
- executeUpdate
  - Statement class, 420
- executeUpdate JDBC method
  - using, 4
- executing SQL statements
  - in applications, 1
- exit codes
  - about, 912
- exports file
  - dblib.def, 442
- external calls
  - creating procedures and functions, 563
- external environments
  - about, 581
  - C API, 587
  - CLR, 584
  - embedded SQL, 587
  - JAVA, 596
  - ODBC, 587
  - PERL, 601
  - PHP, 605
- external function call interface
  - an\_extfn\_api, 568
  - an\_extfn\_result\_set\_column\_data, 573
  - an\_extfn\_result\_set\_column\_info, 572
  - an\_extfn\_result\_set\_info, 571
  - an\_extfn\_value, 570
  - extfn\_cancel method, 566
  - extfn\_post\_load\_library method, 567
  - extfn\_pre\_unload\_library method, 567
  - extfn\_use\_new\_api method, 565
  - get\_piece callback, 574
  - get\_value callback, 574
  - prototype, 565
  - set\_cancel callback, 576
  - set\_value callback, 575
- external function calls
  - about, 563
  - external environment support, 581
- external functions
  - canceling, 576
  - passing parameters, 574
  - prototypes, 565
  - return types, 580
  - unloading libraries, 580
- external objects
  - about, 583
  - comment, 583
  - system table, 582
- external procedure calls
  - about, 563
  - external environment support, 581
- extfn\_cancel
  - external function call interface, 566
- extfn\_post\_load\_library
  - external function call interface, 567
- extfn\_pre\_unload\_library
  - external function call interface, 567
- extfn\_use\_new\_api
  - external function call interface, 565

**F**

- fastLauncherEnabled

- configurable option, 975
  - fat cursors
    - about, 11
  - favorites list
    - configurable options, 975
  - FETCH FOR UPDATE
    - embedded SQL, 30
    - ODBC, 30
  - fetch operation
    - cursors, 10
    - multiple rows, 11
    - scrollable cursors, 11
  - FETCH statement
    - dynamic queries, 468
    - multi-row, 483
    - using, 479
    - using cursors in embedded SQL, 480
    - wide, 483
  - fetchall method
    - Python, 622
  - fetching
    - embedded SQL, 479
    - limits, 9
    - ODBC, 376
  - FieldCount property
    - SADaReader class [SQL Anywhere .NET API], 237
  - file data sources
    - Windows, 942
  - file names
    - .db file extension, 919
    - .log file extension, 919
    - conventions, 918
    - language, 918
    - SQL Anywhere, 919
    - version number, 918
  - file naming conventions
    - about, 918
  - file transfer
    - callback, 512
  - FileDataSourceName property
    - SAConnectionStringBuilder class [SQL Anywhere .NET API], 192
  - files
    - deployment location, 916
    - naming conventions, 918
  - fill\_s\_sqlda function
    - about, 519
  - fill\_sqlda function
    - about, 520
  - fill\_sqlda\_ex function
    - about, 520
  - ForceStart connection parameter
    - db\_start\_engine, 514
  - ForceStart property
    - SAConnectionStringBuilder class [SQL Anywhere .NET API], 192
  - ForeignKeys field
    - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 260
  - free\_filled\_sqlda function
    - about, 521
  - free\_sqlda function
    - about, 522
  - free\_sqlda\_noind function
    - about, 522
  - functions
    - calling DBTools functions, 847
    - embedded SQL, 495
    - external, 563
    - requirements for web clients, 771
    - SQL Anywhere PHP module, 640
    - web clients, 770
- ## G
- get\_piece
    - about, 574
  - get\_value function
    - about, 574
    - using, 579
  - getAutoCommit method
    - JDBC, 418
  - GetBoolean method
    - SADaReader class [SQL Anywhere .NET API], 218
  - GetByte method
    - SADaReader class [SQL Anywhere .NET API], 218
  - GetBytes method
    - SADaReader class [SQL Anywhere .NET API], 219
    - using, 60
  - GetChar method
    - SADaReader class [SQL Anywhere .NET API], 220



---

GetChars method  
     SADaReader class [SQL Anywhere .NET API],  
     220  
     using, 60  
 getConnection method  
     instances, 418  
 GetData method  
     SADaReader class [SQL Anywhere .NET API],  
     221  
 GetDataSources method  
     SADaSourceEnumerator class [SQL  
     Anywhere .NET API], 240  
 GetDataTypeName method  
     SADaReader class [SQL Anywhere .NET API],  
     222  
 GetDateTime method  
     SADaReader class [SQL Anywhere .NET API],  
     222  
 GetDateTimeOffset method  
     SADaReader class [SQL Anywhere .NET API],  
     223  
 GetDecimal method  
     SADaReader class [SQL Anywhere .NET API],  
     223  
 GetDeleteCommand method  
     SACommandBuilder class [SQL Anywhere .NET  
     API], 148  
 GetDouble method  
     SADaReader class [SQL Anywhere .NET API],  
     224  
 GetEnumerator method  
     SADaReader class [SQL Anywhere .NET API],  
     224  
     SAErrorCollection class [SQL Anywhere .NET  
     API], 245  
     SAParameterCollection class [SQL  
     Anywhere .NET API], 286  
 GetFieldType method  
     SADaReader class [SQL Anywhere .NET API],  
     225  
 GetFillParameters method  
     SADaAdapter class [SQL Anywhere .NET API],  
     210  
 GetFloat method  
     SADaReader class [SQL Anywhere .NET API],  
     225  
 GetGuid method  
     SADaReader class [SQL Anywhere .NET API],  
     226  
 GetInsertCommand method  
     SACommandBuilder class [SQL Anywhere .NET  
     API], 150  
 GetInt16 method  
     SADaReader class [SQL Anywhere .NET API],  
     226  
 GetInt32 method  
     SADaReader class [SQL Anywhere .NET API],  
     227  
 GetInt64 method  
     SADaReader class [SQL Anywhere .NET API],  
     227  
 GetKeyword method  
     SAConnectionStringBuilderBase class [SQL  
     Anywhere .NET API], 201  
 GetName method  
     SADaReader class [SQL Anywhere .NET API],  
     228  
 GetObjectData method  
     SAException class [SQL Anywhere .NET API],  
     248  
 GetOrdinal method  
     SADaReader class [SQL Anywhere .NET API],  
     228  
 GetSchema method  
     SAConnection class [SQL Anywhere .NET API],  
     168  
 GetSchemaTable method  
     SADaReader class [SQL Anywhere .NET API],  
     229  
     using SADaReader, 50  
 GetString method  
     SADaReader class, 78  
     SADaReader class [SQL Anywhere .NET API],  
     230  
 GetTimeSpan method  
     SADaReader class [SQL Anywhere .NET API],  
     231  
     using, 60  
 GetUInt16 method  
     SADaReader class [SQL Anywhere .NET API],  
     232  
 GetUInt32 method  
     SADaReader class [SQL Anywhere .NET API],  
     232  
 GetUInt64 method

- SADaReader class [SQL Anywhere .NET API], 232
- GetUpdateCommand method
  - SACommandBuilder class [SQL Anywhere .NET API], 152
- getUpdateCounts
  - BatchUpdateException, 420
- GetUseLongNameAsKeyword method
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 157
  - SAConnectionStringBuilderBase class [SQL Anywhere .NET API], 201
- GetValue method
  - SADaReader class [SQL Anywhere .NET API], 233
- GetValues method
  - SADaReader class [SQL Anywhere .NET API], 234
- GNU compiler
  - embedded SQL support, 441
- GRANT statement
  - JDBC, 427
- H**
- handles
  - about ODBC, 355
  - allocating ODBC, 356
- HasRows property
  - SADaReader class [SQL Anywhere .NET API], 237
- HEADER clause
  - managing, 775
- header files
  - embedded SQL, 441
  - ODBC, 349
- headers
  - accessing in HTTP web services, 745
  - in SOAP web services, 749
- Host
  - accessing HTTP headers, 747
- Host property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 192
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 309
- host variables
  - about, 452
  - data types, 453
  - declaring, 452
  - example, 746
  - not supported in batches, 452
  - SQLDA, 471
  - uses, 457
- HTML services
  - about, 734
  - commenting, 740
  - creating, 736
  - dropping, 739
  - quick start, 768
  - quick start for web clients, 766
  - quick start for web servers, 729
- HTTP headers
  - accessing, 747
- HTTP protocol
  - configuring, 732
  - enabling, 731
- HTTP request headers
  - accessing, 747
  - management, 775
- HTTP requests
  - structures, 801
- HTTP system procedures
  - alphabetical list, 760
- HTTP\_HEADER function
  - example, 746
- HTTP\_VARIABLE function
  - example, 746
- HttpMethod
  - accessing HTTP headers, 748
- HttpQueryString
  - accessing HTTP headers, 748
- HTTPS protocol
  - configuring, 732
  - enabling, 731
- HttpStatus
  - accessing HTTP headers, 748
- HttpURI
  - accessing HTTP headers, 748
- HttpVersion
  - accessing HTTP headers, 748
- I**
- iAnywhere ODBC Driver Manager
  - Unix, 352

---

- iAnywhere.Data.SQLAnywhere
  - Entity Framework support, 40
- iAnywhere.Data.SQLAnywhere namespace
  - SQL Anywhere .NET API reference, 99
- iAnywhere.Data.SQLAnywhere.dll
  - deploying .NET clients, 927
- iAnywhere.Data.SQLAnywhere.dll.config
  - deploying .NET clients, 927
- iAnywhere.Data.SQLAnywhere.gac
  - deploying .NET clients, 927
- iAnywhere.Data.SQLAnywhere.v3.5.dll
  - deploying .NET clients, 927
- iAnywhere.Data.SQLAnywhere.v4.0.dll
  - deploying .NET clients, 927
- iastor registry entry
  - preventing database corruption, 981
- iastorv registry entry
  - preventing database corruption, 981
- identifiers
  - needing quotes, 523
- IdleTimeout property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 193
- import libraries
  - alternatives, 444
  - DBTools, 846
  - embedded SQL, 442
  - introduction, 436
  - ODBC, 349
  - Windows Mobile ODBC, 350
- import statement
  - jConnect, 409
- INCLUDE statement
  - SQLCA, 461
- IndexColumns field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 261
- Indexes field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 261
- IndexOf method
  - SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 121
  - SAParameterCollection class [SQL Anywhere .NET API], 286
- indicator
  - wide fetch, 483
- indicator variables
  - about, 458
  - data type conversion, 460
  - NULL, 459
  - ODBC, 368
  - SQLDA, 471
  - summary of values, 460
  - truncation, 460
- InfoMessage event
  - SACConnection class [SQL Anywhere .NET API], 180
- initialization utility (dbinit)
  - deployment considerations, 989
- InitString property
  - SACConnection class [SQL Anywhere .NET API], 179
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 193
- INOUT parameters
  - Java in the database, 400
- Inprocess option
  - Linked Server, 340
- insensitive cursors
  - about, 21
  - cursor properties, 15
  - delete example, 17
  - embedded SQL, 32
  - introduction, 17
  - update example, 19
- Insert method
  - SAParameterCollection class [SQL Anywhere .NET API], 288
- INSERT statement
  - JDBC, 420
  - multi-row, 483
  - performance, 2
  - wide, 483
  - writing Python scripts, 623
- InsertCommand property
  - SADDataAdapter class [SQL Anywhere .NET API], 210
- InsertDynamic method
  - JDBCExample, 422
- inserts
  - JDBC, 424
- InsertStatic method
  - JDBCExample, 420
- INSTALL JAVA statement
  - using when installing a class, 394

- using when installing a JAR, 395
- install key
  - silent install, 923
- install option
  - SetupVSPackage, 927
- installation
  - registration key, 923
  - silent install, 923
- installation programs
  - deploying, 916
- installer
  - silent install, 923
- installing
  - JAR files into a database, 395
  - Java classes into a database, 393
  - Java classes into a database using SQL, 394
  - Java classes into a database using Sybase Central, 394
  - jConnect metadata support, 409
- Instance field
  - SAFactory class [SQL Anywhere .NET API], 255
- Instance property
  - SADataSourceEnumerator class [SQL Anywhere .NET API], 241
- Integrated property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 193
- Intel storage driver
  - preventing database corruption, 981
- Interactive SQL
  - deploying, 956, 975
  - deploying dbisqlc, 977
  - deploying on Linux and Unix, 967
  - deploying on Windows, 957
  - JDBC escape syntax, 430
  - OEM configurations, 975
  - option settings in deployed applications, 976
  - Unix supported deployment platforms, 967
- Interactive SQL options
  - locking settings in deployed applications, 976
- Interactive SQL utility (dbisql)
  - Unix supported deployment platforms, 967
- interface libraries
  - DBLIB, 435
  - dynamic loading, 444
  - SQL Anywhere C API, 528
- interfaces
  - SQL Anywhere embedded SQL, 435
- Interop
  - web services, 796
- Invalid string or buffer length
  - data alignment, 372
- iODBC
  - driver manager, 354
- IPV6 property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 309
- IsClosed property
  - SADataReader class [SQL Anywhere .NET API], 238
- IsDBNull method
  - SADataReader class [SQL Anywhere .NET API], 235
- IsFixedSize property
  - SAParameterCollection class [SQL Anywhere .NET API], 290
- IsNullable property
  - SAParameter class [SQL Anywhere .NET API], 273
- isolation level
  - JDBC, 418
  - readonly-statement-snapshot, 38
  - snapshot, 38
  - statement-snapshot, 38
- isolation levels
  - ADO programming, 335
  - applications, 37
  - cursor sensitivity, 30
  - cursors, 9
  - DTC, 843
  - lost updates, 29
  - SA\_SQL\_TXN\_READONLY\_STATEMENT\_SNAPSHOT, 374
  - SA\_SQL\_TXN\_SNAPSHOT, 374
  - SA\_SQL\_TXN\_STATEMENT\_SNAPSHOT, 374
  - setting for the SATransaction object, 62
  - SQL\_TXN\_READ\_COMMITTED, 374
  - SQL\_TXN\_READ\_UNCOMMITTED, 374
  - SQL\_TXN\_REPEATABLE\_READ, 374
  - SQL\_TXN\_SERIALIZABLE, 374
- IsolationLevel property
  - SATransaction class [SQL Anywhere .NET API], 315
- ISOLATIONLEVEL\_BROWSE
  - about, 843
- ISOLATIONLEVEL\_CHAOS

---

- about, 843
- ISOLATIONLEVEL\_CURSORSTABILITY
  - about, 843
- ISOLATIONLEVEL\_ISOLATED
  - about, 843
- ISOLATIONLEVEL\_READCOMMITTED
  - about, 843
- ISOLATIONLEVEL\_READUNCOMMITTED
  - about, 843
- ISOLATIONLEVEL\_REPEATABLEREAD
  - about, 843
- ISOLATIONLEVEL\_SERIALIZABLE
  - about, 843
- ISOLATIONLEVEL\_UNSPECIFIED
  - about, 843
- isql.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- IsReadOnly property
  - SAParameterCollection class [SQL Anywhere .NET API], 290
- IsSynchronized property
  - SAParameterCollection class [SQL Anywhere .NET API], 290

## J

- jaccess-1\_4.jar
  - deploying Java-based administration tools, 964
- JAR and ZIP file creation wizard
  - using, 395
- JAR files
  - adding, 395
  - installing, 393
  - installing from Sybase Central, 395
  - installing using SQL, 395
  - updating, 396
  - versions, 396
- Java
  - external environment, 596
  - in the database, 385
  - JDBC, 403
- Java class creation wizard
  - using, 417
- Java classes
  - adding, 394
  - installing, 394

- Java in the database
  - about, 385
  - choosing a Java VM, 389
  - deploying, 979
  - environment variables, 389
  - error handling, 388
  - installing classes, 393
  - Java VM, 387
  - key features, 386
  - main method, 397
  - NoSuchMethodException, 398
  - returning result sets, 398
  - security management, 400
  - starting the VM, 401
  - stopping the VM, 401
  - storing classes, 386
  - supported platforms, 387
  - VM shutdown hooks, 401
- JAVA keyword
  - external environment, 596
- Java Runtime Environment
  - using Java in the database, 389
- Java signatures
  - CREATE PROCEDURE statement [user defined], 598
- Java stored procedures
  - about, 398
  - example, 399
- Java VM
  - JAVA\_HOME environment variable, 389
  - JAVAHOME environment variable, 389
  - selecting, 389
  - shutdown hooks, 401
  - starting, 401
  - stopping, 401
- JAVA\_HOME environment variable
  - starting the Java VM, 389
- java\_location option
  - deprecated, 389
- java\_vm\_options option
  - using, 389
- JavaAccessBridge.dll
  - deploying Java-based administration tools, 964
- JAVAHOME environment variable
  - starting the Java VM, 389
- JAWTAccessBridge.dll
  - deploying Java-based administration tools, 964
- JAX-WS

- installing, 821
- tutorial, 818
- versions, 821
- JAXWS\_HOME
  - environment variable, 821
- JComponents1200.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- jconn3.jar
  - deploying database servers, 979
- jconn4.jar
  - loading jConnect, 410
  - loading jConnect JDBC driver, 415
- jConnect
  - about, 409
  - choosing a JDBC driver, 404
  - CLASSPATH environment variable, 409
  - connecting, 410
  - database setup, 409
  - deploying JDBC clients, 947
  - download, 409
  - external connections, 412
  - installing metadata support, 409
  - loading, 410
  - packages, 409
  - server-side connections, 415
  - system objects, 409
  - URL, 410
  - versions supplied, 409
- JDBC
  - about, 403
  - applications overview, 406
  - autocommit, 418
  - autocommit mode, 35
  - batched inserts, 421
  - batched inserts example, 424
  - client connections, 412
  - client-side, 406
  - connecting to a database, 411
  - connection code, 412
  - connection defaults, 418
  - connections, 406
  - controlling autocommit behavior, 35
  - cursor types, 14
  - data access, 419
  - DELETE statement, 420
  - deploying JDBC clients, 947
  - escape syntax in Interactive SQL, 430
  - example connection, 412
  - example source code, 404
  - INSERT statement, 420
  - introduction to programming, 403
  - jConnect, 409
  - permissions, 427
  - prepared statements, 422
  - requirements, 404
  - result sets, 425
  - server-side, 406
  - server-side connections, 415
  - SQL Anywhere JDBC driver, 407
  - SQL statements, 1
  - transaction isolation level, 418
  - UPDATE statement, 420
  - ways to use, 404
- JDBC callback
  - example, 427
- JDBC drivers
  - choosing, 404
  - compatibility, 404
  - OSGi bundle, 404
  - performance, 404
- JDBC escape syntax
  - using in Interactive SQL, 430
- JDBC-ODBC bridge
  - SQL Anywhere JDBC driver, 404
- JDBCExample class
  - about, 419
- JDBCExample.java
  - about, 419
- jh.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- jlogon.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- jodbc4.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- JRE
  - checking version on Mac OS X, 968
  - using Java in the database, 389
- jre160\_x64

- 
- 64-bit Java Runtime Environment, 959
    - deploying on Windows, 960
  - jre160\_x86
    - 32-bit Java Runtime Environment, 959
      - deploying on Windows, 960
  - jre\_1.6.0\_linux\_sun\_i586
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
  - jre\_1.6.0\_solaris\_sun\_sparc
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
  - js.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - JSON services
    - about, 734
    - commenting, 740
    - creating, 736
    - dropping, 739
    - quick start, 768
    - quick start for web clients, 766
    - quick start for web servers, 729
  - jsyblib610.dll
    - deploying on Windows, 960
  - jsyblib610.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
  - K**
  - Kerberos property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 193
  - Keys property
    - SACConnectionStringBuilderBase class [SQL Anywhere .NET API], 204
  - keyset-driven cursors
    - ODBC, 31
    - value-sensitive, 24
  - L**
  - Language property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 194
  - languages
    - file names, 918
  - LazyClose property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 194
  - LD\_LIBRARY\_PATH environment variable
    - deployment, 918
  - LDAP property
    - SATcpOptionsBuilder class [SQL Anywhere .NET API], 310
  - length SQLDA field
    - about, 471
    - values, 473
  - libdbcapi.dylib
    - PHP support module, 607
  - libdbcapi.so
    - PHP support module, 607
  - libdbcapi\_r.dylib
    - deploying, 985
    - deploying PHP clients, 949
    - PHP support module, 607
  - libdbcapi\_r.so
    - deploying, 985
    - deploying PHP clients, 949
    - PHP support module, 607
  - libdbcis12.dylib
    - deploying database servers, 979
  - libdbcis12.so
    - deploying database servers, 979
  - libdbecc12.so
    - ECC encryption, 987
  - libdbencod12\_r
    - deploying SQL Remote, 991
  - libdbextenv12\_r.dylib
    - deploying, 985
    - PHP support module, 607
  - libdbextenv12\_r.so
    - deploying, 985
    - PHP support module, 607
  - libdbextf.dylib
    - deploying database servers, 979
  - libdbextf.so
    - deploying database servers, 979
  - libdbfile12\_r
    - deploying SQL Remote, 991
  - libdbftp12\_r
    - deploying SQL Remote, 991
  - libdbicu12.dylib
    - deploying JDBC clients, 937
    - deploying ODBC clients, 937

- libdbicu12.so
  - deploying ODBC clients, 937
- libdbicu12\_r.dylib
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
- libdbicu12\_r.so
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
- libdbicudt12.dylib
  - deploying database servers, 979
  - deploying JDBC clients, 937, 948
  - deploying ODBC clients, 937
- libdbicudt12.so
  - deploying database servers, 979
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
- libdbjdbc12.dylib
  - deploying, 985
  - deploying database servers, 979
  - deploying JDBC clients, 948
- libdbjdbc12.so
  - deploying, 985
  - deploying database servers, 979
  - deploying JDBC clients, 948
- libdbjodbc12.dylib
  - deploying database servers, 979
  - deploying on Mac OS X, 968
- libdbjodbc12.so
  - deploying database servers, 979
- libdbjodbc12.so.1
  - deploying on Unix, 968
- libdblib12.dylib
  - deploying database utilities, 989
  - deploying PHP clients, 949
- libdblib12.so
  - deploying database utilities, 989
  - deploying embedded SQL clients, 946
  - deploying PHP clients, 949
- libdblib12\_r
  - deploying SQL Remote, 991
- libdblib12\_r.dylib
  - deploying database utilities, 989
- libdblib12\_r.so
  - deploying on Mac OS X, 968
- libdblib12\_r.so.1
  - deploying on Unix, 968
- libdbodbc12
  - Unix ODBC driver, 351
- libdbodbc12.dylib
  - deploying database servers, 979
  - deploying ODBC clients, 937
- libdbodbc12.so
  - deploying database servers, 979
  - deploying ODBC clients, 937
- libdbodbc12\_n.dylib
  - deploying database servers, 979
  - deploying ODBC clients, 937
- libdbodbc12\_n.so
  - deploying database servers, 979
  - deploying ODBC clients, 937
- libdbodbc12\_r.dylib
  - deploying database servers, 979
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
- libdbodbc12\_r.so
  - deploying database servers, 979
  - deploying ODBC clients, 937
- libdbodbc12\_r.so.1
  - deploying on Unix, 968
- libdbodbcansi12\_r
  - ANSI ODBC driver, 354
- libdbodm12
  - about, 352
  - Unix ODBC driver manager, 352
- libdbodm12.dylib
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
- libdbodm12.so
  - deploying ODBC clients, 937
- libdbodm12.so.1
  - deploying on Unix, 968
- libdbsoftsp\_r.so
  - deploying database utilities, 989
- libdbput12\_r.dylib
  - deploying on Mac OS X, 968
- libdbput12\_r.so.1
  - deploying on Unix, 968
- libdbrsa12.dylib
  - RSA encryption, 987



---

- libdbrsa12.so
  - RSA encryption, 987
- libdbrsa12\_r.dylib
  - RSA encryption, 987
- libdbrsakp12.dylib
  - deploying JDBC clients, 947
  - deploying Open Client, 956
- libdbrsakp12.so
  - deploying JDBC clients, 947
  - deploying Open Client, 956
- libdbrsakp12\_r.dll
  - deploying database servers, 979
- libdbrsakp12\_r.dylib
  - deploying database servers, 979
- libdbscript12\_r.dylib
  - deploying database servers, 979
- libdbscript12\_r.so
  - deploying database servers, 979
- libdbserv12\_r.dylib
  - deploying database servers, 979
- libdbserv12\_r.so
  - AES encryption, 987
  - deploying database servers, 979
- libdbsmtp12\_r
  - deploying SQL Remote, 991
- libdbtasks12.dylib
  - deploying database utilities, 989
  - deploying JDBC clients, 937
- libdbtasks12.so
  - deploying database utilities, 989
  - deploying embedded SQL clients, 946
  - deploying ODBC clients, 937
  - deploying PHP clients, 949
- libdbtasks12\_r
  - deploying SQL Remote, 991
- libdbtasks12\_r.dylib
  - deploying database servers, 979
  - deploying database utilities, 989
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
  - deploying on Mac OS X, 968
- libdbtasks12\_r.so
  - deploying database servers, 979
  - deploying database utilities, 989
  - deploying JDBC clients, 948
  - deploying ODBC clients, 937
- libdbtasks12\_r.so.1
  - deploying on Unix, 968
- libdbtool12\_r
  - about, 845
  - deploying SQL Remote, 991
- libdbtool12\_r.dylib
  - deploying database utilities, 989
  - deploying on Mac OS X, 968
- libdbtool12\_r.so
  - deploying database utilities, 989
- libdbtool12\_r.so.1
  - deploying on Unix, 968
- libjsyblib610\_r.dylib
  - deploying on Mac OS X, 968
- libjsyblib610\_r.so.1
  - deploying on Unix, 968
- libmljodbc12.dylib
  - deploying on Mac OS X, 968
- libmljodbc12.so.1
  - deploying on Unix, 968
- libraries
  - calling external libraries from stored procedures or functions, 563
  - dbtstm.lib, 846
  - dbtool12.lib, 846
  - embedded SQL, 442
  - using the import libraries, 846
- library
  - dblib12.lib, 442
  - dblibtm.lib, 442
  - libdblib12.so, 442
  - libdblib12\_r.so, 442
  - libdbtasks12.so, 442
  - libdbtasks12\_r.so, 442
- library functions
  - embedded SQL, 495
- libsybbr.dll
  - deploying database servers, 979
- libsybbr.dylib
  - deploying database servers, 979
- libsybbr.so
  - deploying database servers, 979
- libulscutil12.so.1
  - deploying on Unix, 968
- licensing
  - web clients, 752
- line length
  - SQL preprocessor output, 437
- line numbers
  - SQL preprocessor utility (sqlpp), 437

- Linked Servers
    - 4-part syntax, 339
    - four-part syntax, 339
    - Inprocess option, 340
    - OLE DB, 339
    - openquery, 339
    - RPC option, 340
    - RPC Out option, 340
    - security context, 340
  - LINQ support
    - .NET data provider features, 40
    - LinqSample, .NET Data Provider sample project, 41
  - LinqSample
    - .NET Data Provider sample project, 41
  - Linux
    - deployment issues, 916
    - directory structure, 916
  - liveness
    - timeout callback, 511
  - LivenessTimeout property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 194
  - LocalOnly property
    - SATcpOptionsBuilder class [SQL Anywhere .NET API], 310
  - lockedPreferences
    - configurable option, 975
  - LogFile property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 194
  - logging
    - web services client information, 802
  - LONG BINARY data type
    - embedded SQL, 487
    - retrieving in embedded SQL, 488
    - sending in embedded SQL, 490
  - LONG NVARCHAR data type
    - embedded SQL, 487
    - retrieving in embedded SQL, 488
    - sending in embedded SQL, 490
  - LONG VARCHAR data type
    - embedded SQL, 487
    - retrieving in embedded SQL, 488
    - sending in embedded SQL, 490
  - LONGBINARY data type
    - embedded SQL, 453
  - LONGNVARCHAR data type
    - embedded SQL, 453
  - LONGVARCHAR data type
    - embedded SQL, 453
  - lost updates
    - about, 28
- ## M
- Mac OS X
    - checking JRE version, 968
    - deployment issues, 916
    - directory structure, 916
  - macros
    - \_SQL\_OS\_WINDOWS, 444
  - main method
    - Java in the database, 397
  - management tools
    - dbtools, 845
  - manual commit mode
    - controlling, 35
    - implementation, 36
    - transactions, 35
  - mapping files
    - deploying to SQL Server, 927
  - maximumDisplayedRows
    - configurable option, 975
  - MaxPoolSize property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 195
  - MDAC
    - deploying, 931
    - version, 931
  - membership
    - result sets, 16
  - MergeModule.CABinet
    - deployment wizard, 920
  - Message Agent (dbremote)
    - deploying SQL Remote, 991
  - Message property
    - SAError class [SQL Anywhere .NET API], 243
    - SAException class [SQL Anywhere .NET API], 249
    - SAInfoMessageEventArgs class [SQL Anywhere .NET API], 257
  - messages
    - callback, 511
    - server, 511
  - MessageType property

---

SAInfoMessageEventArgs class [SQL Anywhere .NET API], 257  
 metadata support  
   installing for jConnect, 409  
 MetaDataCollections field  
   SAMetaDataCollectionNames class [SQL Anywhere .NET API], 262  
 method signatures  
   Java, 598  
 Microsoft SQL Server Management Studio  
   Linked Server, 340  
 Microsoft Transaction Server  
   three-tier computing, 841  
 Microsoft Visual C++  
   embedded SQL support, 441  
 MIME  
   setting types, 744  
 MIME types  
   web services tutorial, 806  
 MinPoolSize property  
   SAConnectionStringBuilder class [SQL Anywhere .NET API], 195  
 mixed cursors  
   ODBC, 31  
 mldesign.jar  
   deploying on Mac OS X, 968  
   deploying on Unix, 968  
   deploying on Windows, 960  
 mljodbc12.dll  
   deploying on Windows, 960  
 mlmon.ini  
   deploying administration tools, 956  
 mlplugin.jar  
   deploying on Mac OS X, 968  
   deploying on Unix, 968  
   deploying on Windows, 960  
 mlstream.jar  
   deploying on Mac OS X, 968  
   deploying on Unix, 968  
   deploying on Windows, 960  
 mobilink.jpr  
   deploying administration tools on Linux/Unix/Mac OS X, 971  
   deploying administration tools on Windows, 961  
 MSDASQL  
   OLE DB provider, 329  
 msixexec  
   about, 922  
   deployment wizard, 919  
   silent install, 923  
 MSSqlToSA.xml  
   deploying .NET clients, 927  
 msxml4.dll  
   deploying UltraLite, 985  
 multi-row fetches  
   ESQL, 483  
 multi-row inserts  
   ESQL, 483  
 multi-row puts  
   ESQL, 483  
 multi-row queries  
   cursors, 480  
 multiple result sets  
   DESCRIBE statement, 493  
   ODBC, 378  
   OEM configurable option, 975  
 multithreaded applications  
   embedded SQL, 463  
   Java in the database, 398  
   multiple SQLCAs in embedded SQL, 465  
   ODBC connections, 360  
   ODBC feature support, 348  
   Unix, 351  
 myDispose method  
   SADataReader class [SQL Anywhere .NET API], 235  
 MyIP property  
   SATcpOptionsBuilder class [SQL Anywhere .NET API], 310

**N**

name SQLDA field  
   about, 471  
 NAMESPACE clause  
   managing, 780  
 namespaces  
   web services, 795  
 NativeError property  
   SAError class [SQL Anywhere .NET API], 243  
   SAException class [SQL Anywhere .NET API], 249  
   SAInfoMessageEventArgs class [SQL Anywhere .NET API], 257  
 NCHAR data type  
   embedded SQL, 453

- Network Service
  - OLE DB, 339
- NewPassword property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 195
- NEXT\_CONNECTION function
  - example, 757
- NEXT\_HTTP\_HEADER function
  - example, 746
- NEXT\_HTTP\_VARIABLE function
  - example, 746
- NextResult method
  - SADataReader class [SQL Anywhere .NET API], 236
- NO SCROLL cursors
  - about, 21
  - embedded SQL, 32
- NodeType property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 195
- NotifyAfter property
  - SABulkCopy class [SQL Anywhere .NET API], 107
- ntodbc.h
  - about, 349
  - compilation platform, 368
- NuGet
  - Entity Framework 4, 64
- NULL
  - dynamic SQL, 469
  - indicator variables, 458
- null-terminated string
  - embedded SQL data type, 449
- NVARCHAR data type
  - embedded SQL, 453
- O**
- objects
  - storage format, 396
- obtaining time values
  - about, 60
- ODBC
  - 64-bit considerations, 368
  - about, 347
  - autocommit mode, 35
  - calling ODBC functions from DllMain, 363
  - conformance, 348
  - connecting with no data source, 945
  - connections for multithreaded applications, 360
  - controlling autocommit behavior, 35
  - cursor support and result sets, 374
  - cursor types, 14
  - cursors, 31
  - data alignment, 372
  - data source types, 942
  - driver deployment, 936
  - driver manager, 348
  - driver name for SQL Anywhere, 357
  - error checking, 382
  - escape syntax, 380
  - external environment, 587
  - FETCH FOR UPDATE, 30
  - handles, 355
  - header files, 349
  - import libraries, 349
  - linking, 349
  - multiple result sets, 378
  - multithreaded applications, 348
  - options, 361
  - prepared statements, 367
  - registry entries, 942
  - requirements, 347
  - result sets, 378
  - sample application, 357
  - sample program, 354
  - SQL statements, 1
  - SQLSetConnectAttr, 361
  - stored procedures, 378
  - Unix development, 351
  - version supported, 348
  - Windows Mobile, 350
- ODBC client
  - deploying, 936
  - install, 936
- ODBC data sources
  - deploying, 942
- ODBC driver managers
  - iODBC, 354
  - Unix, 352
  - unixODBC, 353
  - UTF-32, 354
  - Windows performance, 942
- ODBC drivers
  - Unix, 351
- odbc.h

- 
- about, 349
  - OEM.ini
    - administration tools, 975
  - Offset property
    - SAParameter class [SQL Anywhere .NET API], 273
  - OLE DB
    - about, 329
    - autocommit mode, 35
    - connection parameters, 336
    - connection pooling, 338
    - controlling autocommit behavior, 35
    - cursor types, 14
    - cursors, 31
    - deploying, 930
    - Microsoft Linked Server setup, 339
    - ODBC and, 329
    - provider deployment, 930
    - supported interfaces, 342
    - supported platforms, 330
    - updates, 334
    - updating data through a cursor, 334
  - OLE DB and ADO development
    - about, 329
  - OLE DB and ADO programming interface
    - about, 329
    - introduction, 329
  - OLE transactions
    - three-tier computing, 840
    - three-tier computing terminology, 841
  - online backups
    - embedded SQL, 494
  - Open Client
    - architecture, 721
    - autocommit mode, 35
    - controlling autocommit behavior, 35
    - cursor types, 14
    - data type ranges, 723
    - data types, 723
    - data types compatibility, 723
    - deploying Open Client applications, 956
    - encrypting passwords, 727
    - interface, 721
    - introduction, 721
    - limitations, 727
    - requirements, 722
    - SQL, 724
    - SQL Anywhere limitations, 727
    - SQL statements, 1
  - Open method
    - SACConnection class [SQL Anywhere .NET API], 176
  - OPEN statement
    - using cursors in embedded SQL, 480
  - openquery
    - Linked Server, 339
  - operating systems
    - file names, 918
  - options
    - ODBC applications, 361
    - web services, 761
  - OSGi deployment bundle
    - JDBC 4.0 driver, 404
  - OUT parameters
    - Java in the database, 400
  - overflow errors
    - data type conversion, 723
  - P**
  - packages
    - installing, 395
    - jConnect, 409
  - Padding enumeration [database tools API]
    - description, 863
  - parallel backups
    - db\_backup function, 496
  - parameter size considerations
    - ODBC, 368
  - ParameterName property
    - SAParameter class [SQL Anywhere .NET API], 273
  - parameters
    - substitution, 800
  - Parameters property
    - SACCommand class [SQL Anywhere .NET API], 143
  - passing parameters
    - to external functions, 574
  - Password property
    - SACConnectionStringBuilder class [SQL Anywhere .NET API], 196
  - passwords
    - encrypting in jConnect, 409
    - encrypting in Open Client, 727
    - Interactive SQL, 975

- pdf-transcoder.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- performance
  - cursors, 26
  - cursors and prefetched rows, 27
  - JDBC, 422
  - JDBC drivers, 404
  - prepared statements, 2
- Perl
  - connecting to a database, 615
  - DBD::SQLAnywhere, 611
  - executing SQL statements, 616
  - external environments, 601
  - handling multiple result sets, 617
  - inserting rows, 618
  - installing Perl/DBI support on Unix and Mac OS X, 613
  - installing Perl/DBI support on Windows, 611
  - writing DBD::SQLAnywhere scripts, 614
- Perl DBD::SQLAnywhere
  - about, 611
  - introduction to programming, 611
- Perl DBI module
  - about, 611
- PERL keyword
  - external environment, 601
- perlenv.pl
  - deploying, 986
- permissions
  - JDBC, 427
  - procedures calling external functions, 563
- PersistSecurityInfo property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 196
- personal server
  - deploying, 988
- PHP
  - about, 627
  - API reference, 640
  - configuring PHP, 954
  - data access, 627
  - deploying PHP clients, 949
  - extension, 627
  - external environment, 605
  - installing PHP support on Linux/Solaris, 953
  - installing PHP support on Windows, 952
  - running PHP scripts in web pages, 629
  - versions, 951
  - writing scripts, 630
  - writing web pages, 628
- PHP extension
  - choosing which to use, 951
  - introduction to programming, 627
  - Linux/Solaris, 953
  - testing, 627
  - versions, 951
  - Windows, 952
- PHP functions
  - sasql\_affected\_rows, 642
  - sasql\_close, 643
  - sasql\_commit, 643
  - sasql\_connect, 643
  - sasql\_data\_seek, 644
  - sasql\_disconnect, 645
  - sasql\_error, 645
  - sasql\_errorcode, 646
  - sasql\_escape\_string, 646
  - sasql\_fetch\_array, 647
  - sasql\_fetch\_assoc, 647
  - sasql\_fetch\_field, 648
  - sasql\_fetch\_object, 649
  - sasql\_fetch\_row, 649
  - sasql\_field\_count, 650
  - sasql\_field\_seek, 650
  - sasql\_free\_result, 651
  - sasql\_get\_client\_info, 651
  - sasql\_insert\_id, 652
  - sasql\_message, 652
  - sasql\_multi\_query, 652
  - sasql\_next\_result, 653
  - sasql\_num\_fields, 654
  - sasql\_num\_rows, 654
  - sasql\_pconnect, 655
  - sasql\_prepare, 655
  - sasql\_query, 656
  - sasql\_real\_escape\_string, 657
  - sasql\_real\_query, 657
  - sasql\_result\_all, 658
  - sasql\_rollback, 659
  - sasql\_set\_option, 659
  - sasql\_sqlstate, 671
  - sasql\_stmt\_affected\_rows, 660
  - sasql\_stmt\_bind\_param, 661
  - sasql\_stmt\_bind\_param\_ex, 661

---

- sasql\_stmt\_bind\_result, 662
- sasql\_stmt\_close, 662
- sasql\_stmt\_data\_seek, 663
- sasql\_stmt\_errno, 663
- sasql\_stmt\_error, 664
- sasql\_stmt\_execute, 664
- sasql\_stmt\_fetch, 665
- sasql\_stmt\_field\_count, 665
- sasql\_stmt\_free\_result, 666
- sasql\_stmt\_insert\_id, 666
- sasql\_stmt\_next\_result, 667
- sasql\_stmt\_num\_rows, 667
- sasql\_stmt\_param\_count, 668
- sasql\_stmt\_reset, 668
- sasql\_stmt\_result\_metadata, 669
- sasql\_stmt\_send\_long\_data, 669
- sasql\_stmt\_store\_result, 670
- sasql\_store\_result, 670
- sasql\_use\_result, 671
- PHP hypertext preprocessor
  - about, 627
- PHP keyword
  - external environment, 605
- php-5.1.[1-6]\_sqlanywhere\_extenv12.dll
  - deploying, 986
  - deploying PHP clients, 949
- php-5.1.[1-6]\_sqlanywhere\_extenv12.so
  - deploying, 986
  - deploying PHP clients, 949
- php-5.2.[0-11]\_sqlanywhere\_extenv12.dll
  - deploying, 986
  - deploying PHP clients, 949
- php-5.2.[0-11]\_sqlanywhere\_extenv12.so
  - deploying, 986
  - deploying PHP clients, 949
- php-5.3.[0-5]\_sqlanywhere\_extenv12.dll
  - deploying, 986
  - deploying PHP clients, 949
- php-5.3.[0-5]\_sqlanywhere\_extenv12.so
  - deploying, 986
  - deploying PHP clients, 949
- php.ini
  - configuring PHP, 954
- phpenv.php
  - deploying, 986
  - PHP support module, 607
- PHPRC
  - environment variable, 607
- placeholders
  - dynamic SQL, 466
- platforms
  - cursors, 14
  - Java in the database support, 387
- plug-ins
  - deploying, 957
- policy.12.0.iAnywhere.Data.SQLAnywhere.dll
  - deploying .NET clients, 927
- policy.12.0.iAnywhere.Data.SQLAnywhere.v3.5.dll
  - deploying .NET clients, 927
- policy.12.0.iAnywhere.Data.SQLAnywhere.v4.0.dll
  - deploying .NET clients, 927
- pooling
  - connections with .NET Data Provider, 45
  - web services, 743
- POOLING option
  - .NET Data Provider, 45
- Pooling property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 196
- positioned DELETE statement
  - about, 11
- positioned UPDATE statement
  - about, 11
- positioned updates
  - about, 9
- power failures
  - EnableFlush registry entry, 981
- Precision property
  - SAParameter class [SQL Anywhere .NET API], 274
- prefetch
  - cursor performance, 26
  - cursors, 27
  - fetching multiple rows, 11
- prefetch option
  - cursors, 27
- PrefetchBuffer property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 196
- PrefetchRows property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 197
- Prepare method
  - SACCommand class [SQL Anywhere .NET API], 139
  - using, 3

- PREPARE statement
  - using, 466
- PREPARE TRANSACTION statement
  - and Open Client, 727
- prepared statements
  - ADO.NET overview, 3
  - bind parameters, 3
  - cursors, 7
  - dropping, 3
  - JDBC, 422
  - ODBC, 367
  - Open Client, 725
  - using, 2
- PreparedStatement interface
  - about, 422
- prepareStatement method
  - JDBC, 4
- preparing
  - statements, 2
  - to commit, 841
- preprocessor
  - about, 435
  - running, 437
- primary keys
  - obtaining values for, 57
- ProcedureParameters field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 262
- procedures
  - embedded SQL, 491
  - ODBC, 378
  - requirements for web clients, 771
  - result sets in ESQL, 492
  - web clients, 770
- Procedures field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 263
- program structure
  - embedded SQL, 443
- programming interfaces
  - C API, 527
  - JDBC API, 403
  - ODBC, 347
  - Perl DBD::SQLAnywhere API, 611
  - Python Database API, 619
  - Ruby APIs, 691
  - SQL Anywhere .NET API, 39
  - SQL Anywhere embedded SQL, 435
  - SQL Anywhere OLE DB and ADO APIs, 329
  - SQL Anywhere PHP DBI, 627
  - Sybase Open Client API, 721
- progress messages
  - ODBC, 361
- properties
  - db\_get\_property function, 505
- protocols
  - configuring web services, 732
  - enabling web services, 731
- prototype
  - external functions, 565
- providers
  - SQL Anywhere ASP.NET Health Monitoring Provider, 91
  - SQL Anywhere ASP.NET Membership Provider, 91
  - SQL Anywhere ASP.NET Profiles Provider, 91
  - SQL Anywhere ASP.NET Roles Provider, 91
  - SQL Anywhere ASP.NET Web Parts Personalization Provider, 91
  - supported in .NET, 40
  - supported in ASP.NET, 91
- PUT statement
  - modifying rows through a cursor, 11
  - multi-row, 483
  - wide, 483
- Python
  - closing connections, 621
  - commit method, 623
  - control over type conversion, 624
  - creating connections, 621
  - creating cursors, 622
  - database types, 624
  - executing SQL statements, 622
  - inserting into tables, 623
  - installing Python support on Unix and Mac OS X, 620
  - installing Python support on Windows, 620
  - multiple inserts, 623
  - sqlanydb, 619
  - writing sqlanydb scripts, 621
- Python Database API
  - introduction to programming, 619
- Python Database support
  - about, 619



---

## Q

- qaagent.exe
  - deploying on Windows, 960
- qaconnector.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- qanywhere.jpr
  - deploying administration tools on Linux/Unix/Mac OS X, 971
  - deploying administration tools on Windows, 962
- qaplugin.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- queries
  - ADO Recordset object, 332
  - ADO Recordset object and cursors, 333
  - single-row, 479
- quick start
  - accessing SQL Anywhere web server, 768
  - SQL Anywhere web client, 766
  - SQL Anywhere web server, 729
- quoted identifiers
  - sql\_needs\_quotes function, 523
- QuoteIdentifier method
  - SACommandBuilder class [SQL Anywhere .NET API], 153

## R

- Rails
  - about, 692
  - installing ActiveRecord adapter, 691
  - tutorial, 694
- RAW services
  - about, 734
  - commenting, 740
  - creating, 736
  - dropping, 739
  - quick start, 768
  - quick start for web clients, 766
  - quick start for web servers, 729
- Read method
  - SADataReader class [SQL Anywhere .NET API], 236
- read-only
  - deploying databases, 984
  - read-only cursors
    - about, 15
- READ\_CLIENT\_FILE function
  - ESQL client API callback function, 510
  - ODBC client API callback function, 361
- ReceiveBufferSize property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 310
- record sets
  - ADO programming, 334
- RecordsAffected property
  - SADataReader class [SQL Anywhere .NET API], 238
  - SARowUpdatedEventArgs class [SQL Anywhere .NET API], 301
- Recordset ADO object
  - ADO, 332
  - ADO programming, 335
  - updating data, 334
- Recordset object
  - ADO, 333
- recovery
  - distributed transactions, 844
- reentrant code
  - multithreaded embedded SQL example, 463
- registerDriver method
  - loading SQL Anywhere JDBC 3.0 driver, 407
- registering
  - DLLs for deployment, 984
  - SQL Anywhere .NET Data Provider, 71
  - SQL Anywhere ASP.NET connection string, 93
  - SQL Anywhere ASP.NET providers, 94
- registration key
  - silent install, 923
- registry
  - deploying, 981
  - deploying administration tools on Windows, 965
  - ODBC, 942
  - Wow6432Node, 965
- registry settings
  - SQL Anywhere ODBC driver, 940
- relaysrvr.jpr
  - deploying administration tools on Linux/Unix/Mac OS X, 971
  - deploying administration tools on Windows, 962
- REMOTEPWD
  - using, 411
- Remove method

- SABulkCopyColumnMappingCollection class
  - [SQL Anywhere .NET API], 121
- SAConnectionStringBuilderBase class [SQL Anywhere .NET API], 202
- SAParameterCollection class [SQL Anywhere .NET API], 288
- RemoveAt method
  - SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 122
  - SAParameterCollection class [SQL Anywhere .NET API], 288
- removeShutdownHook
  - Java VM shutdown hooks, 401
- removing
  - ADO.NET data provider, 929
- reportErrors
  - configurable option, 975
- request processing
  - embedded SQL, 494
- requests
  - aborting, 501
- requirements
  - Open Client applications, 722
- ReservedWords field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 263
- ResetCommandTimeout method
  - SACommand class [SQL Anywhere .NET API], 140
- ResetDbType method
  - SAParameter class [SQL Anywhere .NET API], 271
- resource dispensers
  - three-tier computing, 841
- resource managers
  - about, 839
  - three-tier computing, 841
- Restrictions field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 264
- result sets
  - about ADO Recordset object, 332
  - accessing from a web client, 787
  - cursors, 5
  - Java in the database stored procedures, 398
  - JDBC, 425
  - metadata, 33
  - multiple ODBC, 378
  - ODBC, 374
  - ODBC stored procedures, 378
  - Open Client, 726
  - retrieving from a web service, 786
  - retrieving ODBC, 376
  - stored procedures, 492
  - using, 7
  - using ADO Recordset object, 333
- Results method
  - JDBCExample, 425
- retrieving
  - ODBC, 376
  - SQLDA, 477
- RetryConnectionTimeout property
  - SAConnectionStringBuilder class [SQL Anywhere .NET API], 197
- return codes
  - about, 912
  - ODBC, 382
- return types
  - external functions, 580
- return values
  - web clients, 787
- Rollback method
  - SATransaction class [SQL Anywhere .NET API], 313
- ROLLBACK statement
  - cursors, 37
- ROLLBACK TO SAVEPOINT statement
  - cursors, 38
- RollbackTrans ADO method
  - ADO programming, 335
  - updating data, 335
- root web services
  - about, 740
  - web browsing, 762
- RowsCopied property
  - SARowsCopiedEventArgs class [SQL Anywhere .NET API], 299
- RowUpdated event
  - SADataAdapter class [SQL Anywhere .NET API], 213
- RowUpdating event
  - SADataAdapter class [SQL Anywhere .NET API], 213
- RPC option
  - Linked Server, 340
- RPC Out option

---

Linked Server, 340

Ruby

- about, 691
- installing ActiveRecord adapter, 691
- installing native Ruby driver, 691
- installing Ruby/DBI support, 692

Ruby API

- about, 699
- sqlany\_affected\_rows function, 700
- sqlany\_bind\_param function, 700
- sqlany\_clear\_error function, 701
- sqlany\_client\_version function, 701
- sqlany\_commit function, 702
- sqlany\_connect function, 702
- sqlany\_describe\_bind\_param function, 703
- sqlany\_disconnect function, 704
- sqlany\_error function, 704
- sqlany\_execute function, 705
- sqlany\_execute\_direct function, 705
- sqlany\_execute\_immediate function, 706
- sqlany\_fetch\_absolute function, 707
- sqlany\_fetch\_next function, 707
- sqlany\_fini function, 708
- sqlany\_free\_connection function, 709
- sqlany\_free\_stmt function, 709
- sqlany\_get\_bind\_param\_info function, 710
- sqlany\_get\_column function, 710
- sqlany\_get\_column\_info function, 711
- sqlany\_get\_next\_result function, 712
- sqlany\_init function, 713
- sqlany\_new\_connection function, 713
- sqlany\_num\_cols function, 714
- sqlany\_num\_params function, 714
- sqlany\_num\_rows function, 715
- sqlany\_prepare function, 716
- sqlany\_rollback function, 716
- sqlany\_sqlstate function, 717

Ruby APIs

- introduction to programming, 691

Ruby DBI

- about, 695
- connection examples, 696
- installing dbd-sqlanywhere, 692

Ruby on Rails

- about, 692
- installing ActiveRecord adapter, 691
- tutorial, 694

RubyGems

installing, 691

## S

sa\_config.csh

- deployment, 918

sa\_config.sh

- deployment, 918

sa\_external\_library\_unload

- using, 580

SA\_GET\_MESSAGE\_CALLBACK\_PARM

- SQLSetConnectAttr, 361

SA\_REGISTER\_MESSAGE\_CALLBACK

- SQLSetConnectAttr, 361

SA\_REGISTER\_VALIDATE\_FILE\_TRANSFER\_CALLBACK

- SQLSetConnectAttr, 361

sa\_set\_http\_header system procedure

- example, 744

SA\_SQL\_ATTR\_TXN\_ISOLATION

- SQLSetConnectAttr, 361

SA\_SQL\_TXN\_READONLY\_STATEMENT\_SNAPSHOT

- isolation level, 374

SA\_SQL\_TXN\_SNAPSHOT

- isolation level, 374

SA\_SQL\_TXN\_STATEMENT\_SNAPSHOT

- isolation level, 374

SA\_TRANSACTION\_SNAPSHOT

- transaction isolation level, 418

SA\_TRANSACTION\_STATEMENT\_READONLY\_SNAPSHOT

- transaction isolation level, 418

SA\_TRANSACTION\_STATEMENT\_SNAPSHOT

- transaction isolation level, 418

SABulkCopy class [SQL Anywhere .NET API]

- BatchSize property, 106
- BulkCopyTimeout property, 106
- Close method, 103
- ColumnMappings property, 106
- description, 99
- DestinationTableName property, 107
- Dispose method, 103
- NotifyAfter property, 107
- SABulkCopy constructor, 100
- SARowsCopied event, 108
- WriteToServer method, 103

SABulkCopy constructor

- SABulkCopy class [SQL Anywhere .NET API], 100
- SABulkCopyColumnMapping class [SQL Anywhere .NET API]
  - description, 108
  - DestinationColumn property, 112
  - DestinationOrdinal property, 113
  - SABulkCopyColumnMapping constructor, 109
  - SourceColumn property, 113
  - SourceOrdinal property, 114
- SABulkCopyColumnMapping constructor
  - SABulkCopyColumnMapping class [SQL Anywhere .NET API], 109
- SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API]
  - Add method, 117
  - Contains method, 120
  - CopyTo method, 120
  - description, 114
  - IndexOf method, 121
  - Remove method, 121
  - RemoveAt method, 122
  - this property, 122
- SABulkCopyOptions enumeration [SQL Anywhere .NET API]
  - description, 318
- sacapi.h header file
  - SQL Anywhere C API reference, 528
- sacapidll.h
  - interface library, 528
- sacapidll.h header file
  - SQL Anywhere C API reference, 528
- SACommand class
  - about, 46
  - deleting data, 48
  - example, 82
  - inserting data, 48
  - retrieving data, 47
  - updating data, 48
  - using, 47
  - using in a Visual Studio project, 78
  - using prepared statements, 3
- SACommand class [SQL Anywhere .NET API]
  - BeginExecuteNonQuery method, 126
  - BeginExecuteReader method, 128
  - Cancel method, 132
  - CommandText property, 141
  - CommandTimeout property, 141
  - CommandType property, 141
  - Connection property, 142
  - CreateParameter method, 132
  - description, 122
  - DesignTimeVisible property, 142
  - EndExecuteNonQuery method, 133
  - EndExecuteReader method, 135
  - ExecuteNonQuery method, 137
  - ExecuteReader method, 137
  - ExecuteScalar method, 139
  - Parameters property, 143
  - Prepare method, 139
  - ResetCommandTimeout method, 140
  - SACommand constructor, 124
  - Transaction property, 143
  - UpdatedRowSource property, 144
- SACommand constructor
  - SACommand class [SQL Anywhere .NET API], 124
- SACommandBuilder class [SQL Anywhere .NET API]
  - DataAdapter property, 154
  - DeriveParameters method, 148
  - description, 144
  - GetDeleteCommand method, 148
  - GetInsertCommand method, 150
  - GetUpdateCommand method, 152
  - QuoteIdentifier method, 153
  - SACommandBuilder constructor, 147
  - UnquoteIdentifier method, 154
- SACommandBuilder constructor
  - SACommandBuilder class [SQL Anywhere .NET API], 147
- SACommLinksOptionsBuilder class [SQL Anywhere .NET API]
  - All property, 158
  - ConnectionString property, 159
  - description, 155
  - GetUseLongNameAsKeyword method, 157
  - SACommLinksOptionsBuilder constructor, 156
  - SetUseLongNameAsKeyword method, 157
  - SharedMemory property, 159
  - TcpOptionsBuilder property, 159
  - TcpOptionsString property, 160
  - ToString method, 158
- SACommLinksOptionsBuilder constructor
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 156

---

SAConnection class  
     connecting to a database, 43  
     example, 82  
     using in a Visual Studio project, 78

SAConnection class [SQL Anywhere .NET API]  
     BeginTransaction method, 163  
     ChangeDatabase method, 165  
     ChangePassword method, 166  
     ClearAllPools method, 166  
     ClearPool method, 166  
     Close method, 167  
     ConnectionString property, 176  
     ConnectionTimeout property, 177  
     CreateCommand method, 167  
     Database property, 178  
     DataSource property, 178  
     description, 160  
     EnlistDistributedTransaction method, 168  
     EnlistTransaction method, 168  
     GetSchema method, 168  
     InfoMessage event, 180  
     InitString property, 179  
     Open method, 176  
     SAConnection constructor, 162  
     ServerVersion property, 179  
     State property, 179  
     StateChange event, 180

SAConnection constructor  
     SAConnection class [SQL Anywhere .NET API], 162

SAConnection function  
     using in a Visual Studio project, 82

SAConnectionStringBuilder class [SQL Anywhere .NET API]  
     AppInfo property, 186  
     AutoStart property, 186  
     AutoStop property, 187  
     Charset property, 187  
     CommBufferSize property, 187  
     CommLinks property, 187  
     Compress property, 188  
     CompressionThreshold property, 188  
     ConnectionLifetime property, 188  
     ConnectionName property, 188  
     ConnectionPool property, 189  
     ConnectionReset property, 189  
     ConnectionTimeout property, 189  
     DatabaseFile property, 189  
     DatabaseKey property, 190  
     DatabaseName property, 190  
     DatabaseSwitches property, 190  
     DataSourceName property, 190  
     description, 180  
     DisableMultiRowFetch property, 191  
     Elevate property, 191  
     EncryptedPassword property, 191  
     Encryption property, 191  
     Enlist property, 192  
     FileDataSourceName property, 192  
     ForceStart property, 192  
     Host property, 192  
     IdleTimeout property, 193  
     InitString property, 193  
     Integrated property, 193  
     Kerberos property, 193  
     Language property, 194  
     LazyClose property, 194  
     LivenessTimeout property, 194  
     LogFile property, 194  
     MaxPoolSize property, 195  
     MinPoolSize property, 195  
     NewPassword property, 195  
     NodeType property, 195  
     Password property, 196  
     PersistSecurityInfo property, 196  
     Pooling property, 196  
     PrefetchBuffer property, 196  
     PrefetchRows property, 197  
     RetryConnectionTimeout property, 197  
     SAConnectionStringBuilder constructor, 185  
     ServerName property, 197  
     StartLine property, 198  
     Unconditional property, 198  
     UserID property, 198

SAConnectionStringBuilder constructor  
     SAConnectionStringBuilder class [SQL Anywhere .NET API], 185

SAConnectionStringBuilderBase class [SQL Anywhere .NET API]  
     ContainsKey method, 200  
     description, 198  
     GetKeyword method, 201  
     GetUseLongNameAsKeyword method, 201  
     Keys property, 204  
     Remove method, 202  
     SetUseLongNameAsKeyword method, 202

- ShouldSerialize method, 203
- this property, 204
- TryGetValue method, 203
- sacshelp12.chm
  - deploying ODBC clients, 937
- SADDataAdapter
  - obtaining primary key values, 57
- SADDataAdapter class
  - about, 46
  - deleting data, 52
  - inserting data, 52
  - retrieving data, 51
  - updating data, 52
  - using, 51
- SADDataAdapter class [SQL Anywhere .NET API]
  - DeleteCommand property, 210
  - description, 205
  - GetFillParameters method, 210
  - InsertCommand property, 210
  - RowUpdated event, 213
  - RowUpdating event, 213
  - SADDataAdapter constructor, 207
  - SelectCommand property, 211
  - TableMappings property, 211
  - UpdateBatchSize property, 212
  - UpdateCommand property, 212
- SADDataAdapter constructor
  - SADDataAdapter class [SQL Anywhere .NET API], 207
- SADDataReader class
  - example, 83
  - using, 47
  - using in a Visual Studio project, 78
- SADDataReader class [SQL Anywhere .NET API]
  - Close method, 217
  - Depth property, 237
  - description, 214
  - FieldCount property, 237
  - GetBoolean method, 218
  - GetByte method, 218
  - GetBytes method, 219
  - GetChar method, 220
  - GetChars method, 220
  - GetData method, 221
  - GetDataTypeName method, 222
  - GetDateTime method, 222
  - GetDateTimeOffset method, 223
  - GetDecimal method, 223
  - GetDouble method, 224
  - GetEnumerator method, 224
  - GetFieldType method, 225
  - GetFloat method, 225
  - GetGuid method, 226
  - GetInt16 method, 226
  - GetInt32 method, 227
  - GetInt64 method, 227
  - GetName method, 228
  - GetOrdinal method, 228
  - GetSchemaTable method, 229
  - GetString method, 230
  - GetTimeSpan method, 231
  - GetUInt16 method, 232
  - GetUInt32 method, 232
  - GetUInt64 method, 232
  - GetValue method, 233
  - GetValues method, 234
  - HasRows property, 237
  - IsClosed property, 238
  - IsDBNull method, 235
  - myDispose method, 235
  - NextResult method, 236
  - Read method, 236
  - RecordsAffected property, 238
  - this property, 238
- SADDataSourceEnumerator class [SQL Anywhere .NET API]
  - description, 240
  - GetDataSources method, 240
  - Instance property, 241
- SADbType enumeration [SQL Anywhere .NET API]
  - description, 319
- SADbType property
  - SAParameter class [SQL Anywhere .NET API], 274
- SADefault class [SQL Anywhere .NET API]
  - description, 241
  - Value field, 241
- SAError class [SQL Anywhere .NET API]
  - description, 242
  - Message property, 243
  - NativeError property, 243
  - Source property, 243
  - SqlState property, 244
  - ToString method, 242
- SAErrorCollection class [SQL Anywhere .NET API]
  - CopyTo method, 245

---

- Count property, 245
- description, 244
- GetEnumerator method, 245
- this property, 246
- SAException class
  - example, 83
  - using in a Visual Studio project, 79
- SAException class [SQL Anywhere .NET API]
  - description, 246
  - Errors property, 248
  - GetObjectData method, 248
  - Message property, 249
  - NativeError property, 249
  - Source property, 249
- SAFactory class [SQL Anywhere .NET API]
  - CanCreateDataSourceEnumerator property, 254
  - CreateCommand method, 251
  - CreateCommandBuilder method, 251
  - CreateConnection method, 252
  - CreateConnectionStringBuilder method, 252
  - CreateDataAdapter method, 253
  - CreateDataSourceEnumerator method, 253
  - CreateParameter method, 253
  - CreatePermission method, 254
  - description, 249
  - Instance field, 255
- SAInfoMessageEventArgs class [SQL Anywhere .NET API]
  - description, 255
  - Errors property, 256
  - Message property, 257
  - MessageType property, 257
  - NativeError property, 257
  - Source property, 257
  - ToString method, 256
- SAInfoMessageEventHandler delegate [SQL Anywhere .NET API]
  - description, 316
- saip12.jar
  - deploying on Windows, 960
- SAIsolationLevel enumeration [SQL Anywhere .NET API]
  - description, 324
- SAIsolationLevel property
  - SATransaction class [SQL Anywhere .NET API], 316
- sajdbc.jar
  - deploying database servers, 979
  - deploying JDBC clients, 947
  - loading SQL Anywhere JDBC driver, 415
- sajdbc4.jar
  - deploying database servers, 979
  - deploying JDBC clients, 947
- sajvm.jar
  - deploying database servers, 979
- salib.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- salocation option
  - SetupVSPackage, 927
- SAMessageType enumeration [SQL Anywhere .NET API]
  - description, 326
- SAMetaDataCollectionNames class [SQL Anywhere .NET API]
  - Columns field, 259
  - DataSourceInformation field, 260
  - DataTypes field, 260
  - description, 258
  - ForeignKeys field, 260
  - IndexColumns field, 261
  - Indexes field, 261
  - MetaDataCollections field, 262
  - ProcedureParameters field, 262
  - Procedures field, 263
  - ReservedWords field, 263
  - Restrictions field, 264
  - Tables field, 264
  - UserDefinedTypes field, 264
  - Users field, 265
  - ViewColumns field, 265
  - Views field, 266
- samples
  - .NET Data Provider, 75
  - building embedded SQL applications, 445
  - DBTools program, 850
  - dynamic cursors in embedded SQL, 447
  - embedded SQL, 446
  - embedded SQL applications, 445
  - ODBC, 354
  - SimpleViewer, 83
  - static cursors in embedded SQL, 447
- SAOLEDB
  - OLE DB provider, 329
- SAParameter class [SQL Anywhere .NET API]

- DbType property, 272
- description, 266
- Direction property, 272
- IsNullable property, 273
- Offset property, 273
- ParameterName property, 273
- Precision property, 274
- ResetDbType method, 271
- SADDbType property, 274
- SAParameter constructor, 268
- Scale property, 274
- Size property, 275
- SourceColumn property, 275
- SourceColumnNullMapping property, 276
- SourceVersion property, 276
- ToString method, 271
- Value property, 276
- SAParameter constructor
  - SAParameter class [SQL Anywhere .NET API], 268
- SAParameterCollection class [SQL Anywhere .NET API]
  - Add method, 279
  - AddRange method, 283
  - AddWithValue method, 284
  - Clear method, 284
  - Contains method, 284
  - CopyTo method, 285
  - Count property, 289
  - description, 277
  - GetEnumerator method, 286
  - IndexOf method, 286
  - Insert method, 288
  - IsFixedSize property, 290
  - IsReadOnly property, 290
  - IsSynchronized property, 290
  - Remove method, 288
  - RemoveAt method, 288
  - SyncRoot property, 291
  - this property, 291
- SAPermission class [SQL Anywhere .NET API]
  - description, 293
  - SAPermission constructor, 296
- SAPermission constructor
  - SAPermission class [SQL Anywhere .NET API], 296
- SAPermissionAttribute class [SQL Anywhere .NET API]
  - CreatePermission method, 297
  - description, 296
  - SAPermissionAttribute constructor, 297
- SAPermissionAttribute constructor
  - SAPermissionAttribute class [SQL Anywhere .NET API], 297
- saplugin.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- SARowsCopied event
  - SABulkCopy class [SQL Anywhere .NET API], 108
- SARowsCopiedEventArgs class [SQL Anywhere .NET API]
  - Abort property, 299
  - description, 298
  - RowsCopied property, 299
  - SARowsCopiedEventArgs constructor, 298
- SARowsCopiedEventArgs constructor
  - SARowsCopiedEventArgs class [SQL Anywhere .NET API], 298
- SARowsCopiedEventHandler delegate [SQL Anywhere .NET API]
  - description, 317
- SARowUpdatedEventArgs class [SQL Anywhere .NET API]
  - Command property, 301
  - description, 299
  - RecordsAffected property, 301
  - SARowUpdatedEventArgs constructor, 301
- SARowUpdatedEventArgs constructor
  - SARowUpdatedEventArgs class [SQL Anywhere .NET API], 301
- SARowUpdatedEventHandler delegate [SQL Anywhere .NET API]
  - description, 317
- SARowUpdatingEventArgs class [SQL Anywhere .NET API]
  - Command property, 303
  - description, 302
  - SARowUpdatingEventArgs constructor, 303
- SARowUpdatingEventArgs constructor
  - SARowUpdatingEventArgs class [SQL Anywhere .NET API], 303
- SARowUpdatingEventHandler delegate [SQL Anywhere .NET API]
  - description, 317



---

`sasql_affected_rows` function (PHP)  
syntax, 642

`sasql_close` function (PHP)  
syntax, 643

`sasql_commit` function (PHP)  
syntax, 643

`sasql_connect` function (PHP)  
syntax, 643

`sasql_data_seek` function (PHP)  
syntax, 644

`sasql_disconnect` function (PHP)  
syntax, 645

`sasql_error` function (PHP)  
syntax, 645

`sasql_errorcode` function (PHP)  
syntax, 646

`sasql_escape_string` function (PHP)  
syntax, 646

`sasql_fetch_array` function (PHP)  
syntax, 647

`sasql_fetch_assoc` function (PHP)  
syntax, 647

`sasql_fetch_field` function (PHP)  
syntax, 648

`sasql_fetch_object` function (PHP)  
syntax, 649

`sasql_fetch_row` function (PHP)  
syntax, 649

`sasql_field_count` function (PHP)  
syntax, 650

`sasql_field_seek` function (PHP)  
syntax, 650

`sasql_free_result` function (PHP)  
syntax, 651

`sasql_get_client_info` function (PHP)  
syntax, 651

`sasql_insert_id` function (PHP)  
syntax, 652

`sasql_message` function (PHP)  
syntax, 652

`sasql_multi_query` function (PHP)  
syntax, 652

`sasql_next_result` function (PHP)  
syntax, 653

`sasql_num_fields` function (PHP)  
syntax, 654

`sasql_num_rows` function (PHP)  
syntax, 654

`sasql_pconnect` function (PHP)  
syntax, 655

`sasql_prepare` function (PHP)  
syntax, 655

`sasql_query` function (PHP)  
syntax, 656

`sasql_real_escape_string` function (PHP)  
syntax, 657

`sasql_real_query` function (PHP)  
syntax, 657

`sasql_result_all` function (PHP)  
syntax, 658

`sasql_rollback` function (PHP)  
syntax, 659

`sasql_set_option` function (PHP)  
syntax, 659

`sasql_sqlstate` function (PHP)  
syntax, 671

`sasql_stmt_affected_rows` function (PHP)  
syntax, 660

`sasql_stmt_bind_param` function (PHP)  
syntax, 661

`sasql_stmt_bind_param_ex` function (PHP)  
syntax, 661

`sasql_stmt_bind_result` function (PHP)  
syntax, 662

`sasql_stmt_close` function (PHP)  
syntax, 662

`sasql_stmt_data_seek` function (PHP)  
syntax, 663

`sasql_stmt_errno` function (PHP)  
syntax, 663

`sasql_stmt_error` function (PHP)  
syntax, 664

`sasql_stmt_execute` function (PHP)  
syntax, 664

`sasql_stmt_fetch` function (PHP)  
syntax, 665

`sasql_stmt_field_count` function (PHP)  
syntax, 665

`sasql_stmt_free_result` function (PHP)  
syntax, 666

`sasql_stmt_insert_id` function (PHP)  
syntax, 666

`sasql_stmt_next_result` function (PHP)  
syntax, 667

`sasql_stmt_num_rows` function (PHP)  
syntax, 667

- sasql\_stmt\_param\_count function (PHP)
  - syntax, 668
- sasql\_stmt\_reset function (PHP)
  - syntax, 668
- sasql\_stmt\_result\_metadata function (PHP)
  - syntax, 669
- sasql\_stmt\_send\_long\_data function (PHP)
  - syntax, 669
- sasql\_stmt\_store\_result function (PHP)
  - syntax, 670
- sasql\_store\_result function (PHP)
  - syntax, 670
- sasql\_use\_result function (PHP)
  - syntax, 671
- SATcpOptionsBuilder class [SQL Anywhere .NET API]
  - Broadcast property, 308
  - BroadcastListener property, 308
  - ClientPort property, 309
  - description, 304
  - DoBroadcast property, 309
  - Host property, 309
  - IPv6 property, 309
  - LDAP property, 310
  - LocalOnly property, 310
  - MyIP property, 310
  - ReceiveBufferSize property, 310
  - SATcpOptionsBuilder constructor, 307
  - SendBufferSize property, 311
  - ServerPort property, 311
  - TDS property, 311
  - Timeout property, 311
  - ToString method, 308
  - VerifyServerName property, 312
- SATcpOptionsBuilder constructor
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 307
- SAToMSSql10.xml
  - deploying .NET clients, 927
- SATransaction class
  - using, 62
- SATransaction class [SQL Anywhere .NET API]
  - Commit method, 313
  - Connection property, 315
  - description, 312
  - IsolationLevel property, 315
  - Rollback method, 313
  - SAIsolationLevel property, 316
  - Save method, 314
- Save method
  - SATransaction class [SQL Anywhere .NET API], 314
- savepoints
  - cursors, 38
- sbgse2.dll
  - FIPS-certified RSA encryption, 987
- Scale property
  - SAParameter class [SQL Anywhere .NET API], 274
- SCEditor610.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- schema
  - adding the SQL Anywhere ASP.NET provider, 92
  - Health Monitoring provider, 98
  - Membership provider, 96
  - Profile provider, 97
  - Roles provider, 97
  - Web Part Personalization provider, 98
- scjview
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
- scjview.exe
  - deploying on Windows, 960
- scjview.ini
  - deploying administration tools, 956
- SCROLL cursors
  - embedded SQL, 32
  - value-sensitive, 24
- scrollable cursors
  - about, 11
  - JDBC support, 404
- scvwen610.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- security
  - Interactive SQL, 975
  - Java in the database, 400
- security context
  - Linked Server, 340
- security manager
  - about, 400
- SELECT statement
  - single row, 479

---

- using dynamic SELECT statements, 468
- SelectCommand property
  - SADDataAdapter class [SQL Anywhere .NET API], 211
- self-registering DLLs
  - deploying SQL Anywhere, 984
- SendBufferSize property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 311
- sensitive cursors
  - about, 22
  - cursor properties, 15
  - delete example, 17
  - embedded SQL, 32
  - introduction, 17
  - update example, 19
- sensitivity
  - cursors, 17
  - delete example, 17
  - isolation levels, 30
  - SQL Anywhere cursors, 16
  - update example, 19
- serialization
  - objects in tables, 396
- server address
  - embedded SQL function, 505
- Server Explorer
  - Visual Studio, 84
- server side autocommit
  - about, 36
- ServerName property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 197
- ServerPort property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 311
- servers
  - locating from ESQL, 518
  - web, 729
- ServerVersion property
  - SACConnection class [SQL Anywhere .NET API], 179
- services
  - data types, 789
  - web, 729
- SessionCreateTime property
  - about, 756
- SessionID property
  - example, 753
- SessionLastTime property
  - about, 756
- sessions
  - about, 752
  - administering, 757
  - creating, 753
  - deleting, 756
  - detecting, 756
  - errors, 758
- set\_cancel
  - about, 576
- set\_value function
  - about, 575
  - using, 579
- setAutoCommit method
  - about, 418
- setting
  - values using the SQLDA, 476
- setTransactionIsolation method
  - JDBC, 418
- SetupVSPackage
  - deploying .NET clients, 927
  - uninstalling ADO.NET data provider, 929
- SetUseLongNameAsKeyword method
  - SACCommLinksOptionsBuilder class [SQL Anywhere .NET API], 157
  - SACConnectionStringBuilderBase class [SQL Anywhere .NET API], 202
- shared objects
  - calling from stored procedures, 563
  - external procedure calls, 564
- SharedMemory property
  - SACCommLinksOptionsBuilder class [SQL Anywhere .NET API], 159
- ShouldSerialize method
  - SACConnectionStringBuilderBase class [SQL Anywhere .NET API], 203
- showMultipleResultSets
  - configurable option, 975
- showResultsForAllStatements
  - configurable option, 975
- signatures
  - Java methods, 598
- silent install
  - msiexec, 922
- silent installs
  - about, 923

- SimpleCE
  - .NET Data Provider sample project, 41
- SimpleViewer
  - .NET Data Provider sample project, 41
  - .NET project, 83
- SimpleWin32
  - .NET Data Provider sample project, 41
- SimpleXML
  - .NET Data Provider sample project, 41
- single-threaded applications
  - Unix, 351
- Size property
  - SAParameter class [SQL Anywhere .NET API], 275
- snapshot isolation
  - lost updates, 29
  - SQL Anywhere .NET Data Provider, 62
- SOAP
  - supplying variables in an envelope, 784
- SOAP headers
  - management, 777
- SOAP namespace
  - management, 780
- SOAP requests
  - structures, 801
- SOAP services
  - .NET tutorial, 826
  - about, 734
  - commenting, 740
  - creating, 737
  - data types, 789
  - dropping, 739
  - faults, 804
  - JAX-WS tutorial, 818
  - SQL Anywhere web client tutorial, 810
- SOAP system procedures
  - alphabetical list, 760
- SOAPHEADER clause
  - managing, 777
- software
  - return codes, 912
- Source property
  - SAError class [SQL Anywhere .NET API], 243
  - SAException class [SQL Anywhere .NET API], 249
  - SAInfoMessageEventArgs class [SQL Anywhere .NET API], 257
- SourceColumn property
  - SABulkCopyColumnMapping class [SQL Anywhere .NET API], 113
  - SAParameter class [SQL Anywhere .NET API], 275
- SourceColumnNullMapping property
  - SAParameter class [SQL Anywhere .NET API], 276
- SourceOrdinal property
  - SABulkCopyColumnMapping class [SQL Anywhere .NET API], 114
- SourceVersion property
  - SAParameter class [SQL Anywhere .NET API], 276
- sp\_tsql\_environment system procedure
  - setting options for jConnect, 412
- SQL
  - ADO applications, 1
  - applications, 1
  - embedded SQL applications, 1
  - JDBC applications, 1
  - ODBC applications, 1
  - Open Client applications, 1
- SQL Anywhere
  - about, 527
  - C API, 527
- SQL Anywhere .NET API
  - about, 39
  - SABulkCopy class, 99
  - SABulkCopyColumnMapping class, 108
  - SABulkCopyColumnMappingCollection class, 114
  - SABulkCopyOptions enumeration, 318
  - SACommand class, 122
  - SACommandBuilder class, 144
  - SACommLinksOptionsBuilder class, 155
  - SAConnection class, 160
  - SAConnectionStringBuilder class, 180
  - SAConnectionStringBuilderBase class, 198
  - SADDataAdapter class, 205
  - SADDataReader class, 214
  - SADDataSourceEnumerator class, 240
  - SADbType enumeration, 319
  - SADefault class, 241
  - SAError class, 242
  - SAErrorCollection class, 244
  - SAException class, 246
  - SAFactory class, 249
  - SAInfoMessageEventArgs class, 255
  - SAInfoMessageEventHandler delegate, 316

---

SAIsolationLevel enumeration, 324  
 SAMessageType enumeration, 326  
 SAMetaDataCollectionNames class, 258  
 SAParameter class, 266  
 SAParameterCollection class, 277  
 SAPermission class, 293  
 SAPermissionAttribute class, 296  
 SARowsCopiedEventArgs class, 298  
 SARowsCopiedEventHandler delegate, 317  
 SARowUpdatedEventArgs class, 299  
 SARowUpdatedEventHandler delegate, 317  
 SARowUpdatingEventArgs class, 302  
 SARowUpdatingEventHandler delegate, 317  
 SATcpOptionsBuilder class, 304  
 SATransaction class, 312  
 SQL Anywhere .NET API reference  
   iAnywhere.Data.SQLAnywhere namespace, 99  
 SQL Anywhere .NET Data Provider  
   about, 39  
   samples, 75  
 SQL Anywhere 12 Demo.dsn  
   Windows Mobile ODBC, 350  
 SQL Anywhere ASP.NET data providers  
   about, 91  
 SQL Anywhere C API  
   a\_sqlany\_bind\_param structure, 555  
   a\_sqlany\_bind\_param\_info structure, 555  
   a\_sqlany\_column\_info structure, 556  
   a\_sqlany\_data\_direction enumeration, 552  
   a\_sqlany\_data\_info structure, 557  
   a\_sqlany\_data\_type enumeration, 552  
   a\_sqlany\_data\_value structure, 558  
   a\_sqlany\_native\_type enumeration, 553  
   C API, 527  
   external environment, 587  
   interface library, 528  
   sqlany\_affected\_rows method, 528  
   sqlany\_bind\_param method, 529  
   sqlany\_cancel method, 529  
   sqlany\_clear\_error method, 530  
   sqlany\_client\_version method, 530  
   sqlany\_client\_version\_ex method, 530  
   sqlany\_commit method, 531  
   sqlany\_connect method, 531  
   sqlany\_describe\_bind\_param method, 532  
   sqlany\_disconnect method, 533  
   sqlany\_error method, 533  
   sqlany\_execute method, 534  
   sqlany\_execute\_direct method, 535  
   sqlany\_execute\_immediate method, 536  
   sqlany\_fetch\_absolute method, 536  
   sqlany\_fetch\_next method, 537  
   sqlany\_finalize\_interface method, 537  
   sqlany\_fini method, 538  
   sqlany\_fini\_ex method, 538  
   sqlany\_free\_connection method, 539  
   sqlany\_free\_stmt method, 539  
   sqlany\_get\_bind\_param\_info method, 539  
   sqlany\_get\_column method, 540  
   sqlany\_get\_column\_info method, 541  
   sqlany\_get\_data method, 541  
   sqlany\_get\_data\_info method, 542  
   sqlany\_get\_next\_result method, 543  
   sqlany\_init method, 543  
   sqlany\_init\_ex method, 544  
   sqlany\_initialize\_interface method, 545  
   sqlany\_make\_connection method, 545  
   sqlany\_make\_connection\_ex method, 546  
   sqlany\_new\_connection method, 546  
   sqlany\_new\_connection\_ex method, 547  
   sqlany\_num\_cols method, 547  
   sqlany\_num\_params method, 548  
   sqlany\_num\_rows method, 548  
   sqlany\_prepare method, 549  
   sqlany\_reset method, 550  
   sqlany\_rollback method, 550  
   sqlany\_send\_param\_data method, 551  
   sqlany\_sqlstate method, 551  
   SQLAnywhereInterface structure, 558  
 SQL Anywhere C API reference  
   sacapi.h header file, 528  
   sacapidll.h header file, 528  
 SQL Anywhere Database Tools C API reference  
   dbtools.h header file, 852  
 SQL Anywhere JDBC driver  
   about, 403  
   choosing a JDBC driver, 404  
   components, 948  
   connecting, 408  
   deploying JDBC clients, 947  
   loading 3.0 driver, 407  
   loading 4.0 driver, 408  
   required files, 407  
   URL, 408  
   using, 407  
 SQL Anywhere ODBC driver

- ANSI, 354
- components, 936
- customizing name for deployment, 940
- defining data sources, 942
- deploying, 936
- driver name, 357
- install, 936
- linking on Windows, 349
- registry settings, 940
- SQL Anywhere Perl DBD::SQLAnywhere DBI module
  - about, 611
- SQL Anywhere PHP API
  - about, 640
- SQL Anywhere PHP extension
  - about, 627
  - configuring, 954
- SQL Anywhere PHP module
  - API reference, 640
- SQL Anywhere plug-in
  - deployment considerations, 957
- SQL Anywhere Python Database support
  - about, 619
- SQL Anywhere Ruby API
  - functions, 699
- SQL Anywhere web services
  - about, 729
- SQL applications
  - executing SQL statements, 1
- SQL Communications Area
  - about, 461
- SQL preprocessor utility (sqlpp)
  - about, 437
  - running, 437
  - syntax, 437
- SQL Remote
  - deploying, 991
- SQL Server
  - deploying mapping files, 927
- SQL statements
  - executing, 725
  - web clients, 782
  - web services, 742
- SQL/1992
  - SQL preprocessor utility (sqlpp), 437
- SQL/1999
  - SQL preprocessor utility (sqlpp), 437
- SQL/2003
  - SQL preprocessor utility (sqlpp), 437
- SQL/2008
  - SQL preprocessor utility (sqlpp), 437
- SQL\_ATTR\_CONCURRENCY attribute
  - about, 376
- SQL\_ATTR\_CONNECTION\_DEAD
  - SQLGetConnectAttr, 360
- SQL\_ATTR\_CURSOR\_SCROLLABLE attribute
  - about, 376
- SQL\_ATTR\_KEYSET\_SIZE
  - ODBC attribute, 31
- SQL\_ATTR\_MAX\_LENGTH attribute
  - about, 376
- SQL\_ATTR\_ROW\_ARRAY\_SIZE
  - fetching multiple rows, 11
  - ODBC attribute, 31
- SQL\_CALLBACK type declaration
  - about, 510
- SQL\_CALLBACK\_PARM type declaration
  - about, 510
- SQL\_CONCUR\_LOCK
  - concurrency value, 376
- SQL\_CONCUR\_READ\_ONLY
  - concurrency value, 376
- SQL\_CONCUR\_ROWVER
  - concurrency value, 376
- SQL\_CONCUR\_VALUES
  - concurrency value, 376
- SQL\_CURSOR\_KEYSET\_DRIVEN
  - ODBC cursor attribute, 31
- SQL\_ERROR
  - ODBC return code, 382
- SQL\_INVALID\_HANDLE
  - ODBC return code, 382
- SQL\_NEED\_DATA
  - ODBC return code, 382
- sql\_needs\_quotes function
  - about, 523
- SQL\_NO\_DATA\_FOUND
  - ODBC return code, 382
- SQL\_ROWSET\_SIZE
  - fetching multiple rows, 11
- SQL\_SUCCESS
  - ODBC return code, 382
- SQL\_SUCCESS\_WITH\_INFO
  - ODBC return code, 382
- SQL\_TXN\_READ\_COMMITTED
  - isolation level, 374

---

**SQL\_TXN\_READ\_UNCOMMITTED**  
 isolation level, 374

**SQL\_TXN\_REPEATABLE\_READ**  
 isolation level, 374

**SQL\_TXN\_SERIALIZABLE**  
 isolation level, 374

**SQLAllocHandle** ODBC function  
 about, 355  
 binding parameters, 365  
 executing statements, 365  
 using, 356

**sqlany.cvf**  
 deploying database servers, 979

**SQLANY12** environment variable  
 deployment, 918

**sqlany\_affected\_rows** function [Ruby API]  
 description, 700

**sqlany\_affected\_rows** method [SQL Anywhere C API]  
 description, 528

**sqlany\_bind\_param** function [Ruby API]  
 description, 700

**sqlany\_bind\_param** method [SQL Anywhere C API]  
 description, 529

**sqlany\_cancel** method [SQL Anywhere C API]  
 description, 529

**sqlany\_clear\_error** function [Ruby API]  
 description, 701

**sqlany\_clear\_error** method [SQL Anywhere C API]  
 description, 530

**sqlany\_client\_version** function [Ruby API]  
 description, 701

**sqlany\_client\_version** method [SQL Anywhere C API]  
 description, 530

**sqlany\_client\_version\_ex** method [SQL Anywhere C API]  
 description, 530

**sqlany\_commit** function [Ruby API]  
 description, 702

**sqlany\_commit** method [SQL Anywhere C API]  
 description, 531

**sqlany\_connect** function [Ruby API]  
 description, 702

**sqlany\_connect** method [SQL Anywhere C API]  
 description, 531

**sqlany\_describe\_bind\_param** function [Ruby API]  
 description, 703

**sqlany\_describe\_bind\_param** method [SQL Anywhere C API]  
 description, 532

**sqlany\_disconnect** function [Ruby API]  
 description, 704

**sqlany\_disconnect** method [SQL Anywhere C API]  
 description, 533

**sqlany\_error** function [Ruby API]  
 description, 704

**sqlany\_error** method [SQL Anywhere C API]  
 description, 533

**sqlany\_execute** function [Ruby API]  
 description, 705

**sqlany\_execute** method [SQL Anywhere C API]  
 description, 534

**sqlany\_execute\_direct** function [Ruby API]  
 description, 705

**sqlany\_execute\_direct** method [SQL Anywhere C API]  
 description, 535

**sqlany\_execute\_immediate** function [Ruby API]  
 description, 706

**sqlany\_execute\_immediate** method [SQL Anywhere C API]  
 description, 536

**sqlany\_fetch\_absolute** function [Ruby API]  
 description, 707

**sqlany\_fetch\_absolute** method [SQL Anywhere C API]  
 description, 536

**sqlany\_fetch\_next** function  
 about, 707

**sqlany\_fetch\_next** method [SQL Anywhere C API]  
 description, 537

**sqlany\_finalize\_interface** method [SQL Anywhere C API]  
 description, 537

**sqlany\_fini** function [Ruby API]  
 description, 708

**sqlany\_fini** method [SQL Anywhere C API]  
 description, 538

**sqlany\_fini\_ex** method [SQL Anywhere C API]  
 description, 538

**sqlany\_free\_connection** function [Ruby API]  
 description, 709

**sqlany\_free\_connection** method [SQL Anywhere C API]  
 description, 539

- sqlany\_free\_stmt function [Ruby API]
  - description, 709
- sqlany\_free\_stmt method [SQL Anywhere C API]
  - description, 539
- sqlany\_get\_bind\_param\_info function [Ruby API]
  - description, 710
- sqlany\_get\_bind\_param\_info method [SQL Anywhere C API]
  - description, 539
- sqlany\_get\_column function [Ruby API]
  - description, 710
- sqlany\_get\_column method [SQL Anywhere C API]
  - description, 540
- sqlany\_get\_column\_info function [Ruby API]
  - description, 711
- sqlany\_get\_column\_info method [SQL Anywhere C API]
  - description, 541
- sqlany\_get\_data method [SQL Anywhere C API]
  - description, 541
- sqlany\_get\_data\_info method [SQL Anywhere C API]
  - description, 542
- sqlany\_get\_next\_result function [Ruby API]
  - description, 712
- sqlany\_get\_next\_result method [SQL Anywhere C API]
  - description, 543
- sqlany\_init function [Ruby API]
  - description, 713
- sqlany\_init method [SQL Anywhere C API]
  - description, 543
- sqlany\_init\_ex method [SQL Anywhere C API]
  - description, 544
- sqlany\_initialize\_interface method [SQL Anywhere C API]
  - description, 545
- sqlany\_make\_connection method [SQL Anywhere C API]
  - description, 545
- sqlany\_make\_connection\_ex method [SQL Anywhere C API]
  - description, 546
- sqlany\_new\_connection function [Ruby API]
  - description, 713
- sqlany\_new\_connection method [SQL Anywhere C API]
  - description, 546
- sqlany\_new\_connection\_ex method [SQL Anywhere C API]
  - description, 547
- sqlany\_num\_cols function [Ruby API]
  - description, 714
- sqlany\_num\_cols method [SQL Anywhere C API]
  - description, 547
- sqlany\_num\_params function [Ruby API]
  - description, 714
- sqlany\_num\_params method [SQL Anywhere C API]
  - description, 548
- sqlany\_num\_rows function [Ruby API]
  - description, 715
- sqlany\_num\_rows method [SQL Anywhere C API]
  - description, 548
- sqlany\_prepare function [Ruby API]
  - description, 716
- sqlany\_prepare method [SQL Anywhere C API]
  - description, 549
- sqlany\_reset method [SQL Anywhere C API]
  - description, 550
- sqlany\_rollback function [Ruby API]
  - description, 716
- sqlany\_rollback method [SQL Anywhere C API]
  - description, 550
- sqlany\_send\_param\_data method [SQL Anywhere C API]
  - description, 551
- sqlany\_sqlstate function [Ruby API]
  - description, 717
- sqlany\_sqlstate method [SQL Anywhere C API]
  - description, 551
- sqlanydb
  - about, 619
  - installing on Unix and Mac OS X, 620
  - installing on Windows, 620
  - Python Database API, 619
  - writing Python scripts, 621
- SQLANYDIR
  - msiexec, 922
- sqlanywhere.jpr
  - deploying administration tools on Linux/Unix/Mac OS X, 970
  - deploying administration tools on Windows, 961
- sqlanywhere\_commit function (deprecated)
  - syntax, 672
- sqlanywhere\_connect function (deprecated)
  - syntax, 672



---

sqlanywhere\_data\_seek function (deprecated)  
     syntax, 673

sqlanywhere\_disconnect function (deprecated)  
     syntax, 674

sqlanywhere\_en12.chm  
     deploying on Windows, 978

sqlanywhere\_en12.map  
     deploying on Windows, 978

sqlanywhere\_error function (deprecated)  
     syntax, 675

sqlanywhere\_errorcode function (deprecated)  
     syntax, 675

sqlanywhere\_execute function (deprecated)  
     syntax, 676

sqlanywhere\_fetch\_array function (deprecated)  
     syntax, 677

sqlanywhere\_fetch\_field function (deprecated)  
     syntax, 678

sqlanywhere\_fetch\_object function (deprecated)  
     syntax, 679

sqlanywhere\_fetch\_row function (deprecated)  
     syntax, 680

sqlanywhere\_free\_result function (deprecated)  
     syntax, 681

sqlanywhere\_identity function (deprecated)  
     syntax, 681

sqlanywhere\_insert\_id function (deprecated)  
     syntax, 681

sqlanywhere\_num\_fields function (deprecated)  
     syntax, 682

sqlanywhere\_num\_rows function (deprecated)  
     syntax, 683

sqlanywhere\_pconnect function (deprecated)  
     syntax, 684

sqlanywhere\_query function (deprecated)  
     syntax, 684

sqlanywhere\_result\_all function (deprecated)  
     syntax, 685

sqlanywhere\_rollback function (deprecated)  
     syntax, 687

sqlanywhere\_set\_option function (deprecated)  
     syntax, 687

SQLAnywhereInterface structure [SQL Anywhere C API]  
     description, 558

SQLBindCol ODBC function  
     about, 376  
     parameter size, 368  
     storage alignment, 372

SQLBindParam ODBC function  
     parameter size, 368

SQLBindParameter function  
     ODBC prepared statements, 4  
     prepared statements, 367

SQLBindParameter ODBC function  
     about, 365  
     parameter size, 368  
     storage alignment, 372  
     stored procedures, 378

SQLBrowseConnect ODBC function  
     about, 357

SQLBulkOperations  
     ODBC function, 11

SQLCA  
     about, 461  
     changing, 463  
     fields, 461  
     length of, 461  
     managing multiple, 465  
     threads, 463

sqlcabc SQLCA field  
     about, 461

sqlcaid SQLCA field  
     about, 461

sqlcode SQLCA field  
     about, 461

SQLColAttribute ODBC function  
     parameter size, 368

SQLColAttributes ODBC function  
     parameter size, 368

SQLConnect ODBC function  
     about, 357  
     Windows performance, 942

SQLCOUNT  
     sqlerror SQLCA field element, 462

sqllda SQLDA field  
     about, 471

SQLDA  
     about, 469  
     allocating, 495  
     descriptors, 34  
     dynamic SQL, 466  
     fields, 471  
     filling using fill\_sqlda, 520  
     filling using fill\_sqlda\_ex, 520  
     freeing, 519

- host variables, 471
- sqlen field, 473
- strings and fill\_sqlda, 520
- strings and fill\_sqlda\_ex, 520
- sqlda\_storage function
  - about, 523
- sqlda\_string\_length function
  - about, 524
- sqldabc SQLDA field
  - about, 471
- sqldaaid SQLDA field
  - about, 471
- sqldata SQLDA field
  - about, 471
- SQLDATETIME data type
  - embedded SQL, 453
- sqldef.h
  - data types, 449
  - software exit codes location, 913
- SQLDescribeCol ODBC function
  - parameter size, 368
- SQLDescribeParam ODBC function
  - parameter size, 368
- SQLDriverConnect ODBC function
  - about, 357
- sqlerrd SQLCA field
  - about, 462
- sqlerrmc SQLCA field
  - about, 462
- sqlerrml SQLCA field
  - about, 462
- SQLError ODBC function
  - about, 382
- sqlerror SQLCA field
  - elements, 462
  - SQLCOUNT, 462
  - SQLIOCOUNT, 462
  - SQLIOESTIMATE, 463
- sqlerror\_message function
  - about, 524
- sqlerrp SQLCA field
  - about, 462
- SQLExecDirect ODBC function
  - about, 365
  - bound parameters, 365
- SQLExecute function
  - ODBC prepared statements, 4
- SQLExtendedFetch
  - fetching multiple rows, 11
  - ODBC function, 10
- SQLExtendedFetch ODBC function
  - about, 376
  - parameter size, 368
  - stored procedures, 378
- SQLFetch
  - ODBC function, 10
- SQLFetch ODBC function
  - about, 376
  - stored procedures, 378
- SQLFetchScroll
  - fetching multiple rows, 11
  - ODBC function, 10
- SQLFetchScroll ODBC function
  - about, 376
  - parameter size, 368
- SQLFreeHandle ODBC function
  - using, 356
- SQLFreeStmt function
  - ODBC prepared statements, 4
- SQLGetConnectAttr ODBC function
  - about, 360
- SQLGetData ODBC function
  - about, 376
  - parameter size, 368
  - storage alignment, 372
- SQLGetDescRec ODBC function
  - parameter size, 368
- sqlind SQLDA field
  - about, 471
- SQLIOCOUNT
  - sqlerror SQLCA field element, 462
- SQLIOESTIMATE
  - sqlerror SQLCA field element, 463
- SQLJ standard
  - about, 385
- sqlen SQLDA field
  - about, 471
  - DESCRIBE statement, 473
  - describing values, 473
  - retrieving values, 477
  - sending values, 476
  - values, 473
- SQLLEN versus SQLINTEGER
  - ODBC, 368
- sqlname SQLDA field
  - about, 471

---

SQLNumResultCols ODBC function  
     stored procedures, 378

SQLParamOptions ODBC function  
     parameter size, 368

sqlpp utility  
     about, 437  
     preprocessor options, 437  
     supported compilers, 441  
     syntax, 437

SQLPrepare function  
     about, 367  
     ODBC prepared statements, 4

SQLPutData ODBC function  
     parameter size, 368

SQLRETURN  
     ODBC return code type, 382

SQLRowCount ODBC function  
     parameter size, 368

SQLSetConnectAttr  
     ODBC applications, 361

SQLSetConnectAttr ODBC function  
     about, 360  
     transaction isolation levels, 374

SQLSetConnectOption ODBC function  
     parameter size, 368

SQLSetDescRec ODBC function  
     parameter size, 368

SQLSetParam ODBC function  
     parameter size, 368

SQLSetPos ODBC function  
     about, 377  
     parameter size, 368

SQLSetScrollOptions ODBC function  
     parameter size, 368

SQLSetStmtAttr ODBC function  
     cursor characteristics, 375

SQLSetStmtOption ODBC function  
     parameter size, 368

SqlState property  
     SAError class [SQL Anywhere .NET API], 244

sqlstate SQLCA field  
     about, 462

SQLTransact ODBC function  
     about, 357

sqltype SQLDA field  
     about, 471  
     DESCRIBE statement, 473

SQLULEN versus SQLINTEGER  
     ODBC, 368

sqlvar SQLDA field  
     about, 471  
     contents, 471

sqlwarn SQLCA field  
     about, 462

SSDLToSA12.tt  
     deploying .NET 4.0 clients, 927

SSIS  
     deploying mapping files, 927

standards  
     SQLJ, 385

starting  
     databases using jConnect, 411

StartLine property  
     SAConnectionStringBuilder class [SQL  
     Anywhere .NET API], 198

State property  
     .NET Data Provider, 46  
     SAConnection class [SQL Anywhere .NET API],  
     179

StateChange event  
     SAConnection class [SQL Anywhere .NET API],  
     180

statement handles  
     ODBC, 355

statements  
     insert, 2

static cursors  
     about, 21  
     ODBC, 31

static SQL  
     about, 466

stax-api-1.0.jar  
     deploying on Mac OS X, 968  
     deploying on Unix, 968  
     deploying on Windows, 960

stored functions  
     calling external functions, 563

stored procedures  
     calling external functions, 563  
     creating in embedded SQL, 491  
     executing in embedded SQL, 491  
     INOUT parameters and Java, 400  
     Java in the database, 398  
     OUT parameters and Java, 400  
     result sets in ESQL, 492  
     SQL Anywhere .NET Data Provider, 61

- string
  - data type, 524
- strings
  - blank padding of DT\_NSTRING, 449
  - blank padding of DT\_STRING, 449
- strlen\_or\_ind
  - ODBC, 368
- structure packing
  - header files, 441
- supported platforms
  - OLE DB, 330
- Sybase Central
  - adding JAR files, 395
  - adding Java classes, 394
  - adding ZIP files, 395
  - configuring for deployment, 975
  - deploying, 956
  - deploying administration tools on Linux/Unix/Mac OS X, 970
  - deploying administration tools on Windows, 961
  - deploying on Linux and Unix, 967
  - deploying on Windows, 957
  - deployment considerations, 957
  - installing jConnect metadata support, 409
- Sybase Open Client support
  - about, 721
- sybasecentral610.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- symlink
  - deployment on Unix, 917
- SyncRoot property
  - SAPParameterCollection class [SQL Anywhere .NET API], 291
- system data sources
  - Windows, 942
- system procedures
  - HTTP, 760
  - SOAP, 760
- system requirements
  - SQL Anywhere .NET Data Provider, 69
- System.Transactions
  - using, 62
- T**
- table adapter
  - Visual Studio, 88
- TableMappings property
  - SADDataAdapter class [SQL Anywhere .NET API], 211
- Tables field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 264
- TableViewer
  - .NET Data Provider sample project, 41
- TcpOptionsBuilder property
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 159
- TcpOptionsString property
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 160
- TDS property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 311
- temporary options
  - setting in ODBC applications, 361
- this property
  - SABulkCopyColumnMappingCollection class [SQL Anywhere .NET API], 122
  - SACConnectionStringBuilderBase class [SQL Anywhere .NET API], 204
  - SADDataReader class [SQL Anywhere .NET API], 238
  - SAErrorCollection class [SQL Anywhere .NET API], 246
  - SAPParameterCollection class [SQL Anywhere .NET API], 291
- threaded applications
  - Unix, 917
- threads
  - Java in the database, 398
  - multiple SQLCAs, 465
  - multiple thread management in embedded SQL, 463
  - ODBC, 348
  - ODBC applications, 360
  - Unix development, 351
- three-tier computing
  - about, 839
  - architecture, 839
  - Distributed Transaction Coordinator, 841
  - distributed transactions, 840
  - EAServer, 841
  - Microsoft Transaction Server, 841

---

- resource dispensers, 841
- resource managers, 841
- Time structure
  - time values in .NET Data Provider, 60
- Timeout property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 311
- times
  - obtaining with .NET Data Provider, 60
- TimeSpan
  - SQL Anywhere .NET Data Provider, 60
- TIMESTAMP data type
  - conversion, 723
- ToString method
  - SACommLinksOptionsBuilder class [SQL Anywhere .NET API], 158
  - SAError class [SQL Anywhere .NET API], 242
  - SAInfoMessageEventArgs class [SQL Anywhere .NET API], 256
  - SAParameter class [SQL Anywhere .NET API], 271
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 308
- tracing
  - .NET support, 72
- transaction processing
  - using the SQL Anywhere .NET Data Provider, 62
- Transaction property
  - SACommand class [SQL Anywhere .NET API], 143
- transactions
  - ADO, 335
  - application development, 34
  - autocommit mode, 35
  - choosing ODBC transaction isolation level, 374
  - controlling autocommit behavior, 35
  - cursors, 37
  - distributed, 839
  - isolation level, 37
  - ODBC, 357
  - OLE DB, 335
  - using distributed, 843
- TransactionScope class
  - using, 62
- troubleshooting
  - cursor positioning, 9
  - Java in the database methods, 398
- truncation
  - FETCH statement, 460
  - indicator variables, 460
  - on FETCH, 460
- TryGetValue method
  - SACConnectionStringBuilderBase class [SQL Anywhere .NET API], 203
- tutorials
  - developing a .NET database application, 83
  - using JAX-WS to access a SOAP/DISH web service, 818
  - using MIME types in a web client, 806
  - using SQL Anywhere to access a SOAP service, 810
  - using the .NET Data Provider Simple code sample, 76
  - using the .NET Data Provider Table Viewer code sample, 79
  - using Visual C# to access a SOAP/DISH web service, 826
- two-phase commit
  - and Open Client, 727
  - managing distributed transactions, 841
  - three-tier computing, 840
- TYPE clause
  - example, 784
  - specifying, 773

## U

- ulplugin.jar
  - deploying on Windows, 960
- ulscutil12.dll
  - deploying on Windows, 960
- ultralite.jpr
  - deploying administration tools on Linux/Unix/Mac OS X, 972
  - deploying administration tools on Windows, 962
- unchained mode
  - controlling, 35
  - implementation, 36
  - transactions, 35
- Unconditional property
  - SACConnectionStringBuilder class [SQL Anywhere .NET API], 198
- Unicode
  - linking ODBC applications for Windows Mobile, 351
- uninstall

---

- ADO.NET data provider, 929
  - uninstall option
    - SetupVSPackage, 929
  - uninstallall option
    - SetupVSPackage, 929
  - unique cursors
    - about, 15
  - Unit enumeration [database tools API]
    - description, 864
  - Unix
    - deployment issues, 916
    - directory structure, 916
    - multithreaded applications, 917
    - ODBC, 351
    - ODBC driver managers, 352
  - unixODBC
    - driver manager, 353
  - unixodbc.h
    - about, 349
    - compilation platform, 368
  - Unload enumeration [database tools API]
    - description, 864
  - unload utility (dbunload)
    - deployment considerations, 989
    - deployment for version 9 and earlier databases, 990
  - unmanaged code
    - dbdata.dll, 71
  - UnquoteIdentifier method
    - SACommandBuilder class [SQL Anywhere .NET API], 154
  - UNSIGNED BIGINT data type
    - embedded SQL, 453
  - UPDATE statement
    - JDBC, 420
    - positioned, 11
  - UpdateBatch ADO method
    - ADO programming, 335
    - updating data, 335
  - UpdateBatchSize property
    - SADDataAdapter class [SQL Anywhere .NET API], 212
  - UpdateCommand property
    - SADDataAdapter class [SQL Anywhere .NET API], 212
  - UpdatedRowSource property
    - SACommand class [SQL Anywhere .NET API], 144
  - updates
    - cursor, 334
    - upgrade database wizard
      - installing jConnect metadata support, 409
    - upgrade utility (dbupgrad)
      - installing jConnect metadata support, 409
    - URL clause
      - specifying, 771
    - URLs
      - database, 411
      - interpreting, 762
      - jConnect, 410
      - session management, 754
      - SQL Anywhere JDBC driver, 408
      - supplying variables, 783
    - user data sources
      - Windows, 942
    - User-Agent
      - accessing HTTP headers, 747
    - UserDefinedTypes field
      - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 264
    - UserID property
      - SAConnectionStringBuilder class [SQL Anywhere .NET API], 198
    - UserList enumeration [database tools API]
      - description, 865
    - Users field
      - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 265
    - UTF-32
      - ODBC driver managers, 354
    - utilities
      - deploying database utilities, 989
      - return codes, 912
      - SQL preprocessor (sqlpp) syntax, 437
- ## V
- Validation enumeration [database tools API]
    - description, 865
  - Value field
    - SADefault class [SQL Anywhere .NET API], 241
  - Value property
    - SAParameter class [SQL Anywhere .NET API], 276
  - value-sensitive cursors
    - about, 24
    - delete example, 17

---

- introduction, 17
- update example, 19
- VARCHAR data type
  - embedded SQL, 453
- variables
  - accessing in HTTP web services, 745
  - in SOAP web services, 749
  - supplying to HTTP web services, 783
- velocity-dep.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- velocity.jar
  - deploying on Mac OS X, 968
  - deploying on Unix, 968
  - deploying on Windows, 960
- Verbosity enumeration [database tools API]
  - description, 866
- VerifyServerName property
  - SATcpOptionsBuilder class [SQL Anywhere .NET API], 312
- Version enumeration [database tools API]
  - description, 867
- version number
  - file names, 918
- ViewColumns field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 265
- Views field
  - SAMetaDataCollectionNames class [SQL Anywhere .NET API], 266
- visible changes
  - cursors, 16
- Visual Basic
  - support in .NET Data Provider, 39
  - tutorial, 84
- Visual C#
  - tutorial, 84
- Visual C++
  - embedded SQL support, 441
- VM
  - Java VM, 387
  - shutdown hooks, 401
  - starting Java, 401
  - stopping Java, 401

## W

- web clients
  - about, 766
  - accessing result sets, 787
  - function and procedure requirements, 771
  - functions and procedures, 770
  - managing HTTP request headers, 775
  - managing SOAP headers, 777
  - managing SOAP namespace, 780
  - quick start, 766
  - retrieving result sets, 786
  - specifying ports, 774
  - specifying request types, 773
  - SQL statements, 782
  - substitution parameters, 800
  - supplying variables, 783
  - URL clause, 771
- web pages
  - adding PHP scripts to, 628
  - customizing, 744
  - running PHP scripts in, 629
- web servers
  - about, 729
  - application development, 744
  - configuring protocols, 732
  - enabling protocols, 731
  - PHP API, 627
  - quick start, 729
  - starting, 731
  - starting multiple, 733
- web services
  - about, 729
  - accessing, 766
  - accessing HTTP variables and headers, 745
  - accessing result sets, 787
  - accessing SOAP headers, 749
  - alphabetical list of system procedures, 760
  - altering, 736
  - array types, 796
  - character sets, 759
  - client log file, about, 802
  - commenting, 740
  - configuring protocols, 732
  - connection properties, 761
  - creating, 736
  - data types, 789
  - developing, 744

- dropping, 739
  - enabling protocols, 731
  - errors, 804
  - host variables, 746
  - HTTP\_HEADER example, 746
  - HTTP\_VARIABLE example, 746
  - interpreting URLs, 762
  - list of web services-related system procedures, 760
  - logging, 802
  - maintaining, 735
  - managing, 734
  - managing sessions, 752
  - MIME types tutorial, 806
  - NEXT\_HTTP\_HEADER example, 746
  - NEXT\_HTTP\_VARIABLE example, 746
  - options, 761
  - pooling, 743
  - references, 803
  - retrieving result sets, 786
  - root, 740
  - SOAP data types, 795
  - SOAP/DISH tutorial, 810
  - SQL statements, 742
  - structure types, 796
  - types, 734
  - wide fetches
    - about, 11
    - ESQL, 483
  - wide inserts
    - ESQL, 483
    - JDBC, 424
  - wide puts
    - ESQL, 483
  - Windows
    - deploying administration tools, 957
    - OLE DB support, 329
    - silent install, 922
  - Windows Mobile
    - dbtool12.dll, 845
    - Java in the database unsupported, 387
    - ODBC, 350
    - OLE DB support, 329
  - WindowsAccessBridge.dll
    - deploying Java-based administration tools, 964
  - WITH HOLD clause
    - cursors, 9
  - work tables
    - cursor performance, 26
  - WRITE\_CLIENT\_FILE function
    - ESQL client API callback function, 510
    - ODBC client API callback function, 361
  - WriteToServer method
    - SABulkCopy class [SQL Anywhere .NET API], 103
  - WSDL compiler
    - about, 803
  - WSDLC
    - about, 803
  - wsimport
    - JAX-WS and web services, 822
  - wstx-asl-3.2.6.jar
    - deploying on Mac OS X, 968
    - deploying on Unix, 968
    - deploying on Windows, 960
- ## X
- XML services
    - about, 734
    - commenting, 740
    - creating, 736
    - dropping, 739
    - quick start, 768
    - quick start for web clients, 766
    - quick start for web servers, 729
  - XMLCONCAT function
    - example, 745
  - XMLELEMENT function
    - example, 745