Copyright and trademarks

Copyright © 2009 iAnywhere Solutions, Inc. Portions copyright © 2009 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 ........................................................................................................... xi

About the SQL Anywhere documentation .............................................................. xii

Introduction to Programming with SQL Anywhere ............................. 1

SQL Anywhere data access programming interfaces .................................................. 3
  SQL Anywhere .NET support ..................................................................... 4
  SQL Anywhere OLE DB and ADO support .................................................. 5
  SQL Anywhere ODBC support ................................................................. 6
  SQL Anywhere JDBC support ................................................................... 7
  SQL Anywhere embedded SQL ................................................................... 8
  SQL Anywhere C language support ........................................................... 9
  SQL Anywhere Perl DBI support .............................................................. 10
  SQL Anywhere Python support .................................................................. 11
  SQL Anywhere PHP support ..................................................................... 12
  SQL Anywhere Ruby support ..................................................................... 13
  SQL Anywhere web services support ......................................................... 14
  Sybase Open Client support ......................................................................... 15

SQL Anywhere Explorer ......................................................................................... 17
  Introduction to the SQL Anywhere Explorer ............................................... 18
  Using the SQL Anywhere Explorer ............................................................ 19

Using SQL in applications ..................................................................................... 23
  Executing SQL statements in applications ................................................. 24
  Preparing statements ................................................................................. 26
  Introduction to cursors ................................................................................. 29
  Working with cursors ................................................................................... 32
  Choosing cursor types ................................................................................ 39
  SQL Anywhere cursors ................................................................................. 41
  Describing result sets ................................................................................. 58
  Controlling transactions in applications ....................................................... 60

Three-tier computing and distributed transactions ............................................. 65
  Introduction to three-tier computing and distributed transactions .......... 66
  Three-tier computing architecture ............................................................. 67
Using distributed transactions ................................................................. 70
Using EAServer with SQL Anywhere ........................................................ 72

Java in the database .................................................................................. 75
Java support in SQL Anywhere ................................................................. 77
Introduction to Java support ...................................................................... 78
Java in the database Q & A ....................................................................... 80
Java error handling ................................................................................... 84
The runtime environment for Java in the database ................................. 85
Creating a Java class for use with SQL Anywhere ................................. 89
Choosing a Java VM .................................................................................. 91
Install the sample Java class .................................................................... 93
Using the CLASSPATH variable ............................................................... 94
Accessing methods in the Java class ....................................................... 95
Accessing fields and methods of the Java object ..................................... 96
Installing Java classes into a database .................................................. 98
Special features of Java classes in the database ..................................... 102
Starting and stopping the Java VM ......................................................... 106

SQL Anywhere Data Access APIs ......................................................... 107
SQL Anywhere .NET Data Provider .......................................................... 109
SQL Anywhere .NET Data Provider features ........................................ 110
Running the sample projects ................................................................. 111
Using the .NET Data Provider in a Visual Studio project ...................... 112
Connecting to a database ....................................................................... 114
Accessing and manipulating data ........................................................... 117
Using stored procedures ....................................................................... 134
Transaction processing .......................................................................... 136
Error handling and the SQL Anywhere .NET Data Provider .................. 138
Deploying the SQL Anywhere .NET Data Provider ............................... 139
Tracing support ....................................................................................... 141
Tutorial: Using the SQL Anywhere .NET Data Provider ........................ 145
Introduction to the .NET Data Provider tutorial .................................... 146
Using the Simple code sample ............................................................... 147
Using the jConnect JDBC driver ................................................................. 507
Connecting from a JDBC client application ................................................. 511
Using JDBC to access data ........................................................................ 517
Using JDBC escape syntax ........................................................................ 526
iAnywhere JDBC 3.0 API support ............................................................... 529
SQL Anywhere embedded SQL ................................................................. 531
Introduction to embedded SQL ................................................................... 532
Sample embedded SQL programs .............................................................. 538
Embedded SQL data types ......................................................................... 542
Using host variables ................................................................................. 546
The SQL Communication Area (SQLCA) .................................................... 555
Static and dynamic SQL ........................................................................... 561
The SQL descriptor area (SQLDA) ............................................................... 565
Fetching data ............................................................................................. 574
Sending and retrieving long values ............................................................... 582
Using simple stored procedures ................................................................. 586
Embedded SQL programming techniques ............................................... 589
SQL preprocessor .................................................................................... 590
Library function reference ......................................................................... 593
Embedded SQL statement summary ......................................................... 615
SQL Anywhere C API reference ................................................................ 617
Introduction to the SQL Anywhere C API version 1.0 ............................... 618
sacapidll.h .............................................................................................. 619
sacapi.h .................................................................................................. 621
a_sqlany_bind_param struct ..................................................................... 639
a_sqlany_bind_param_info struct ............................................................... 640
a_sqlany_column_info struct .................................................................... 641
a_sqlany_data_info struct ........................................................................ 642
a_sqlany_data_value struct ...................................................................... 643
SQLAnywhereInterface struct .................................................................... 644
a_sqlany_data_direction enumeration ...................................................... 647
a_sqlany_data_type enumeration .............................................................. 648
a_sqlany_native_type enumeration ............................................................ 650
sacapi_error_size constant ....................................................................... 651
sqlany_current_api_version constant ......................................................... 652
SQL Anywhere C API examples ................................................................. 653
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to the database tools interface</td>
<td>948</td>
</tr>
<tr>
<td>Using the database tools interface</td>
<td>950</td>
</tr>
<tr>
<td>DBTools functions</td>
<td>957</td>
</tr>
<tr>
<td>DBTools structures</td>
<td>967</td>
</tr>
<tr>
<td>DBTools enumeration types</td>
<td>1007</td>
</tr>
<tr>
<td>Exit codes</td>
<td>1013</td>
</tr>
<tr>
<td>Software component exit codes</td>
<td>1014</td>
</tr>
<tr>
<td>Deploying SQL Anywhere</td>
<td>1015</td>
</tr>
<tr>
<td>Deploying databases and applications</td>
<td>1017</td>
</tr>
<tr>
<td>Introduction to deployment</td>
<td>1018</td>
</tr>
<tr>
<td>Understanding installation directories and file names</td>
<td>1020</td>
</tr>
<tr>
<td>Using the <strong>Deployment Wizard</strong></td>
<td>1023</td>
</tr>
<tr>
<td>Using a silent install for deployment</td>
<td>1027</td>
</tr>
<tr>
<td>Deploying client applications</td>
<td>1029</td>
</tr>
<tr>
<td>Deploying administration tools</td>
<td>1048</td>
</tr>
<tr>
<td>Deploying database servers</td>
<td>1074</td>
</tr>
<tr>
<td>Deploying external environment support</td>
<td>1079</td>
</tr>
<tr>
<td>Deploying security</td>
<td>1081</td>
</tr>
<tr>
<td>Deploying embedded database applications</td>
<td>1082</td>
</tr>
<tr>
<td>Glossary</td>
<td>1087</td>
</tr>
<tr>
<td>Index</td>
<td>1119</td>
</tr>
</tbody>
</table>
About this book

Subject

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

Audience

This book is intended for application developers writing programs that work directly with one of the SQL Anywhere interfaces.
About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats that contain identical information.

- **HTML Help**  The online Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools.

  If you are using a Microsoft Windows operating system, the online Help is provided in HTML Help (CHM) format. To access the documentation, choose **Start» Programs» SQL Anywhere 11 » Documentation » Online Books**.

  The administration tools use the same online documentation for their Help features.

- **Eclipse**  On Unix platforms, the complete online Help is provided in Eclipse format. To access the documentation, run `sadoc` from the `bin32` or `bin64` directory of your SQL Anywhere 11 installation.

- **DocCommentXchange**  DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation.

  Use DocCommentXchange to:
  
  ○ View documentation
  ○ Check for clarifications users have made to sections of documentation
  ○ Provide suggestions and corrections to improve documentation for all users in future releases

  Visit [http://dcx.sybase.com](http://dcx.sybase.com).

- **PDF**  The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information. To download Adobe Reader, visit [http://get.adobe.com/reader/](http://get.adobe.com/reader/).

  To access the PDF documentation on Microsoft Windows operating systems, choose **Start» Programs» SQL Anywhere 11 » Documentation » Online Books - PDF Format**.

  To access the PDF documentation on Unix operating systems, use a web browser to open `install-dir/documentation/en/pdf/index.html`.

About the books in the documentation set

The SQL Anywhere documentation consists of the following books:

- **SQL Anywhere 11 - Introduction**  This book introduces SQL Anywhere 11, a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.

- **SQL Anywhere 11 - Changes and Upgrading**  This book describes new features in SQL Anywhere 11 and in previous versions of the software.

- **SQL Anywhere Server - Database Administration**  This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database
files, backup procedures, security, high availability, replication with the Replication Server, and administration utilities and options.

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

- **SQL Anywhere Server - SQL Reference**  This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).

- **SQL Anywhere Server - SQL Usage**  This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.

- **MobiLink - Getting Started**  This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.

- **MobiLink - Client Administration**  This book describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases. This book also describes the Dbmlsync API, which allows you to integrate synchronization seamlessly into your C++ or .NET client applications.

- **MobiLink - Server Administration**  This book describes how to set up and administer MobiLink applications.

- **MobiLink - Server-Initiated Synchronization**  This book describes MobiLink server-initiated synchronization, a feature that allows the MobiLink server to initiate synchronization or perform actions on remote devices.

- **QAnywhere**  This book describes QAnywhere, which is a messaging platform for mobile, wireless, desktop, and laptop clients.

- **SQL Remote**  This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.

- **UltraLite - Database Management and Reference**  This book introduces the UltraLite database system for small devices.

- **UltraLite - C and C++ Programming**  This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.

- **UltraLite - M-Business Anywhere Programming**  This book describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows Mobile, or Windows.

- **UltraLite -.NET Programming**  This book describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.

- **UltraLiteJ**  This book describes UltraLiteJ. With UltraLiteJ, you can develop and deploy database applications in environments that support Java. UltraLiteJ supports BlackBerry smartphones and Java SE environments. UltraLiteJ is based on the iAnywhere UltraLite database product.
Error Messages

This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. In most cases, the software behaves the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows**  The Microsoft Windows family includes Windows Vista and Windows XP, used primarily on server, desktop, and laptop computers, and Windows Mobile used on mobile devices.
  
  Unless otherwise specified, when the documentation refers to Windows, it refers to all Windows-based platforms, including Windows Mobile.

- **Unix**  Unless otherwise specified, when the documentation refers to Unix, it refers to all Unix-based platforms, including Linux and Mac OS X.

Directory and file names

In most cases, references to directory and file names are similar on all supported platforms, with simple transformations between the various forms. In these cases, Windows conventions are used. Where the details are more complex, the documentation shows all relevant forms.

These are the conventions used to simplify the documentation of directory and file names:

- **Uppercase and lowercase directory names**  On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

  On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

  On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

  The documentation uses the Windows forms of directory names. In most cases, you can convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names**  The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir\Documentation\en PDF* (Windows form).
On Unix, replace the backslash with the forward slash. The PDF documentation is found in install-dir/documentation/en/pdf.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as .exe or .bat. On Unix, executable file names have no suffix.

  For example, on Windows, the network database server is dbsrv11.exe. On Unix, it is dbsrv11.

- **install-dir** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY11 is created and refers to this location. The documentation refers to this location as install-dir.

  For example, the documentation may refer to the file install-dir\readme.txt. On Windows, this is equivalent to %SQLANY11%\readme.txt. On Unix, this is equivalent to $SQLANY11/readme.txt or $[SQLANY11]/readme.txt.

  For more information about the default location of install-dir, see “SQLANY11 environment variable” [SQL Anywhere Server - Database Administration].

- **samples-dir** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable SQLANYSAMP11 is created and refers to this location. The documentation refers to this location as samples-dir.

  To open a Windows Explorer window in samples-dir, from the Start menu, choose Programs » SQL Anywhere 11 » Sample Applications And Projects.

  For more information about the default location of samples-dir, see “SQLANYSAMP11 environment variable” [SQL Anywhere Server - Database Administration].

**Command prompts and command shell syntax**

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

  
  $x
tcip(host=127.0.0.1)

  Where parentheses cause syntax problems, substitute curly braces:

  
  $x
tcip{host=127.0.0.1}
If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

```
-x "tcpip(host=127.0.0.1)"
```

- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

```
-ek "my \"secret\" key"
```

In many shells, the value of the key would be `my "secret" key`.

- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax `%ENVVAR%`. In Unix shells, environment variables are specified using the syntax `$ENVVAR` or `${ENVVAR}`.

---

**Graphic icons**

The following icons are used in this documentation.

- A client application.

![Client Application Icon]

- A database server, such as Sybase SQL Anywhere.

![Database Server Icon]

- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.

![Database Icon]

- Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.

![Replication Icon]
Contacting the documentation team

We would like to receive your opinions, suggestions, and feedback on this Help.

To submit your comments and suggestions, send an email to the SQL Anywhere documentation team at iasdoc@sybase.com. Although we do not reply to emails, your feedback helps us to improve our documentation, so your input is welcome.

DocCommentXchange

You can also leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases


Finding out more and requesting technical support

Additional information and resources are available at the Sybase iAnywhere Developer Community at http://www.sybase.com/developer/library/sql-anywhere-techcorner.

If you have questions or need help, you can post messages to the Sybase iAnywhere newsgroups listed below.

When you write to one of these newsgroups, always provide details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: dbeng11 -v.

The newsgroups are located on the forums.sybase.com news server.
The newsgroups include the following:

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

For web development issues, see http://groups.google.com/group/sql-anywhere-web-development.

**Newsgroup disclaimer**

iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Technical Advisors, and other staff, assist on the newsgroup service when they have time. They offer their help on a volunteer basis and may not be available regularly to provide solutions and information. Their ability to help is based on their workload.
Introduction to Programming with SQL Anywhere

This section introduces you to programming with SQL Anywhere,

SQL Anywhere data access programming interfaces ................................................................. 3
SQL Anywhere Explorer ............................................................................................................ 17
Using SQL in applications ....................................................................................................... 23
Three-tier computing and distributed transactions ................................................................. 65
SQL Anywhere data access programming interfaces

Contents

SQL Anywhere .NET support .................................................................................................. 4
SQL Anywhere OLE DB and ADO support ........................................................................ 5
SQL Anywhere ODBC support .......................................................................................... 6
SQL Anywhere JDBC support ............................................................................................ 7
SQL Anywhere embedded SQL ......................................................................................... 8
SQL Anywhere C language support .................................................................................. 9
SQL Anywhere Perl DBI support ...................................................................................... 10
SQL Anywhere Python support ......................................................................................... 11
SQL Anywhere PHP support ............................................................................................. 12
SQL Anywhere Ruby support ............................................................................................ 13
SQL Anywhere web services support ................................................................................ 14
Sybase Open Client support ........................................................................................... 15
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.

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” on page 109
- “iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)” on page 178
- “Tutorial: Using the SQL Anywhere .NET Data Provider” on page 145
SQL Anywhere OLE DB and ADO support

SQL Anywhere includes an OLE DB provider for OLE DB and ADO programmers.

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. For more information, see “SQL Anywhere .NET support” on page 4.

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, see “SQL Anywhere OLE DB and ADO development” on page 447.
**SQL Anywhere ODBC support**

ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft. 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 with the exception of Windows Mobile. It is provided as a library for Unix.

The primary documentation for ODBC is the Microsoft ODBC Software Development Kit.

**ODBC applications**

You can develop applications using a variety of development tools and programming languages, as shown in the figure below, and accessing the SQL Anywhere database server using the ODBC API.

For example, of the applications supplied with SQL Anywhere, InfoMaker and PowerDesigner Physical Data Model use ODBC to connect to the database.

**See also**

- “SQL Anywhere ODBC API” on page 463
SQL Anywhere JDBC support

JDBC is a call-level interface for Java applications. Developed by Sun Microsystems, JDBC provides Java programmers 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 a pure Java JDBC driver, named jConnect. It also includes the iAnywhere JDBC driver, which is a type 2 driver. Both are described in “SQL Anywhere JDBC driver” on page 501.

For information about choosing a driver, see “Choosing a JDBC driver” on page 502.

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.

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.

See also

- “Choosing a JDBC driver” on page 502
- “SQL Anywhere JDBC driver” on page 501
SQL Anywhere 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.

See also

- “SQL Anywhere embedded SQL” on page 531
SQL Anywhere C language support

The SQL Anywhere C 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.

See also

● “SQL Anywhere C API reference” on page 617
SQL Anywhere 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.

See also

- “SQL Anywhere Perl DBD::SQLAnywhere DBI module” on page 721
SQL Anywhere Python support

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.

See also

- “SQL Anywhere Python Database support” on page 731
SQL Anywhere PHP support

PHP provides the ability to retrieve information from many popular databases. SQL Anywhere includes a module that provides access to SQL Anywhere databases from PHP. You can use the PHP language to retrieve information from SQL Anywhere databases and provide dynamic web content on your own websites.

The SQL Anywhere PHP module 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.

See also

- “SQL Anywhere PHP API” on page 739
SQL Anywhere Ruby 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.

See also

- “SQL Anywhere for Ruby” on page 809
SQL Anywhere web services support

SQL Anywhere web services provide client applications an alternative to data access APIs such as JDBC and ODBC. Web services can be accessed from client applications written in a variety of languages and running on a variety of platforms. Even common scripting languages such as Perl and Python can provide access to web services.

See also

- “SQL Anywhere web services” on page 849
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, such as Replication Server.

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 or Replication Server, use Open Client. The Open Client API is also supported by Sybase Adaptive Server Enterprise.

See also

- “Using SQL Anywhere as an Open Server” [SQL Anywhere Server - Database Administration]
SQL Anywhere Explorer

Contents

Introduction to the SQL Anywhere Explorer .............................................................. 18
Using the SQL Anywhere Explorer ............................................................................. 19
Introduction to the SQL Anywhere Explorer

The SQL Anywhere Explorer is a component that lets you connect to SQL Anywhere and UltraLite databases from Visual Studio.

For information about using the SQL Anywhere Explorer for UltraLite, see “SQL Anywhere Explorer for UltraLite” [UltraLite - .NET Programming].
Using the SQL Anywhere Explorer

In Visual Studio, you can use the SQL Anywhere Explorer to create connections to SQL Anywhere databases. Once you connect to a database, you can:

- view the tables, views, and procedures in the database
- view the data stored in tables and views
- design programs to open connections with the SQL Anywhere database, or to retrieve and manipulate data
- drag and drop database objects onto C# or Visual Basic code or forms so that the IDE automatically generates code that references the selected object

You can also open Sybase Central and Interactive SQL from Visual Studio by choosing the corresponding command from the **Tools** menu.

**Installation note**

If you install SQL Anywhere software on a Windows computer that already has Visual Studio installed, the installation process detects the presence of Visual Studio and performs the necessary integration steps. If you install Visual Studio after installing SQL Anywhere, or install a new version of Visual Studio, the process to integrate SQL Anywhere with Visual Studio must be performed at a command prompt as follows:

- Ensure Visual Studio is not running.
- At a command prompt, run `install-dir\Assembly\v2\SetupVSPackage.exe /install`.

**Working with database connections in Visual Studio**

Use the SQL Anywhere Explorer to display the SQL Anywhere database connections under the **Data Connections** node. You must create a data connection to view the data in the tables and views.

You can list database tables, views, stored procedures, and functions in the SQL Anywhere Explorer and expand individual tables to list their columns. The properties for an object selected in the SQL Anywhere Explorer window appear in the Visual Studio **Properties** pane.

**To add a SQL Anywhere database connection in Visual Studio**

1. Open the SQL Anywhere Explorer by choosing **View » SQL Anywhere Explorer**.
2. In the SQL Anywhere Explorer window, right-click **Data Connections**, and then choose **Add Connection**.
3. Select **SQL Anywhere**, and then click **OK**.
4. Enter the appropriate values to connect to your database.
5. Click **OK**.
A connection is made to the database, and the connection is added to the Data Connections list.

**To remove a SQL Anywhere database connection from Visual Studio**

1. Open the SQL Anywhere Explorer by choosing View » SQL Anywhere Explorer.
2. In the SQL Anywhere Explorer window, right-click the data connections you want to remove, and then choose Delete.

The connection is removed from the SQL Anywhere Explorer window.

**Configuring the SQL Anywhere Explorer**

The Visual Studio Options window includes settings that you can use to configure the SQL Anywhere Explorer.

**To access SQL Anywhere Explorer options**

1. From the Visual Studio Tools menu, choose Options.
2. In the left pane of the Options window, expand SQL Anywhere.
3. Click General to configure the SQL Anywhere Explorer general options as required.

- **Limit Query Results Sent To Output Window** Specify the number of rows that appear in the Output window. The default value is 500.
- **Show System Objects In Server Explorer** Check this option if you want to see system objects in the Microsoft Server Explorer. This is not a SQL Anywhere Explorer option but a Server Explorer option. System objects include those owned by the "dbo" user.
- **Sort Objects** Choose to sort objects in the SQL Anywhere Explorer window by object name or by object owner name.
- **Generate UI Code When Dropping A Table Or View Onto The Designer** Generate the code for tables or views that you drag and drop onto the Windows forms designer.
- **Generate Insert, Update, And Delete Commands For Data Adapters** Generate INSERT, UPDATE, and DELETE commands for the data adapter when you drag and drop a table or view onto a C# or Visual Basic document.
- **Generate Table Mappings For Data Adapters** Generate table mappings for the data adapter when you drag and drop a table onto a C# or Visual Basic document.

**Adding database objects using the SQL Anywhere Explorer**

In Visual Studio, when you drag certain database objects from the SQL Anywhere Explorer and drop them onto Visual Studio designers, the IDE automatically creates new components that reference the selected objects. You can configure the settings for drag and drop operations by choosing Tools » Options in Visual Studio and opening the SQL Anywhere node.
For example, if you drag a stored procedure from the SQL Anywhere Explorer onto a Windows form, the IDE automatically creates a Command object preconfigured to call that stored procedure.

The following table lists the objects you can drag from the SQL Anywhere Explorer, and describes the components created when you drop them onto a Visual Studio Forms Designer or Code Editor.

<table>
<thead>
<tr>
<th>Item</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data connection</td>
<td>Creates a data connection.</td>
</tr>
<tr>
<td>Table</td>
<td>Creates an adapter.</td>
</tr>
<tr>
<td>View</td>
<td>Creates an adapter.</td>
</tr>
<tr>
<td>Stored procedure or function</td>
<td>Creates a command.</td>
</tr>
</tbody>
</table>

To create a new data component using the SQL Anywhere Explorer

1. Open the form or class that you want to add a data component to.
2. In the SQL Anywhere Explorer, select the object you want to use.
3. Drag the object from the SQL Anywhere Explorer to the Forms Designer or Code Editor.

Working with tables using the SQL Anywhere Explorer

The SQL Anywhere Explorer enables you to view the properties and data for tables and views in a SQL Anywhere database from within Visual Studio.

To view a table or view data in Visual Studio

1. Connect to a SQL Anywhere database using the SQL Anywhere Explorer.
2. In the SQL Anywhere Explorer window, expand your database, and then expand Tables or Views, depending on the object you want to view.
3. Right-click a table or view, and then choose Retrieve Data.

The data in the selected table or view appears in the Output window in Visual Studio.

Working with procedures and functions using the SQL Anywhere Explorer

If changes are made to a stored procedure, you can refresh the procedure from the SQL Anywhere Explorer to get the latest changes to columns or parameters.
To refresh a procedure in Visual Studio

1. Connect to a SQL Anywhere database.
2. Right-click the procedure and choose **Refresh**.

The parameters and columns are updated if any changes have been made to the procedure in the database.
Using SQL in applications

Contents

Executing SQL statements in applications ................................................................. 24
Preparing statements .................................................................................................. 26
Introduction to cursors ................................................................................................. 29
Working with cursors .................................................................................................... 32
Choosing cursor types ................................................................................................. 39
SQL Anywhere cursors ................................................................................................ 41
Describing result sets ................................................................................................. 58
Controlling transactions in applications ....................................................................... 60
Executing SQL statements 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:

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

  See “SQL Anywhere .NET Data Provider” on page 109.

- **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:

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

  See “SQL Anywhere ODBC API” on page 463.

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

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

  See “SQL Anywhere JDBC driver” on page 501.

- **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';
  ```

  See “SQL Anywhere embedded SQL” on page 531.

- **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);
  ```

  See “Sybase Open Client API” on page 839.

- **Application development tools** Application development tools such as the members of the Sybase Enterprise Application Studio family provide their own SQL objects, which use either ODBC (PowerBuilder) or JDBC (Power J) internally.
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.

For more information about stored procedures and triggers, see “Using procedures, triggers, and batches” [*SQL Anywhere Server - SQL Usage*].

Java classes in the database can use the JDBC interface in the same way as Java applications outside the server. This chapter discusses some aspects of JDBC. For more information about using JDBC, see “SQL Anywhere JDBC driver” on page 501.
Preparing 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 find yourself using 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.

For information about cursors, see “Introduction to cursors” on page 29.

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.
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.

To 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.

To use a prepared statement (ADO.NET)

1. Create an SACommand object holding the statement.
   ```c#
   SACommand cmd = new SACommand(
       "SELECT * FROM Employees WHERE Surname=?", conn );
   ```
2. Declare data types for any parameters in the statement.
   Use the SACommand.CreateParameter method.
3. Prepare the statement using the Prepare method.
   ```c#
   cmd.Prepare();
   ```
4. Execute the statement.
   ```c#
   SADataReader reader = cmd.ExecuteReader();
   ```

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

To 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 `samples-dir\SQLAnywhere\ODBCPrepare`.
For more information about ODBC prepared statements, see the ODBC SDK documentation, and “Executing prepared statements” on page 483.

**To 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 `set<`*Type*`>` 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 `samples-dir\SQLAnywhere\JDBC\JDBCExample.java`.

   For more information about using prepared statements in JDBC, see “Using prepared statements for more efficient access” on page 519.

**To 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].

**To 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 “Using SQL in Open Client applications” on page 844.
Introduction to cursors

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 39.

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 39.

What are 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.
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.

To 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 SACommand.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.

To 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 “Preparing statements” on page 26.

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].
In the case of 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 574.

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 574.

Prefetching rows
In some cases, the interface library may undertake performance optimizations internally (such as prefetching results), so these steps in the client application may not correspond exactly to software operations.

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.
Cursor positioning problems
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. In some cases, 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. See “Use work tables in query processing (use All-rows optimization goal)” [SQL Anywhere Server - SQL Usage].
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.

Configuring cursors on opening
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. See “isolation_level option [database] [compatibility]” [SQL Anywhere Server - Database Administration].

- **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.

Fetching rows 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.

To 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 SDataReader 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 SDataReader.NextResult method. See “NextResult method” on page 340.

- **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 “Working with result sets” on page 491.
● **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 523.

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

For more information about using cursors in embedded SQL, see “Using cursors in embedded SQL” on page 575.

● **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 “Using cursors” on page 844.

**Fetching multiple rows**

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.

**Fetching with 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.
Modifying rows through a cursor

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 modification 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. See “FROM clause” [SQL Anywhere Server - SQL Reference].
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.
Understanding 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.

For more information, see “DECLARE statement” [SQL Anywhere Server - SQL Reference], and “FOR statement” [SQL Anywhere Server - SQL Reference].

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.
Canceling cursor operations

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

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.
Choosing cursor types

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

For information about SQL Anywhere cursors, see “SQL Anywhere cursors” on page 41.

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.
  For more information, see “Working with result sets” on page 491.
- Embedded SQL supports all types of cursors.
- For JDBC:
  ○ The iAnywhere JDBC driver supports the JDBC 2.0 and JDBC 3.0 specifications and permits the declaration of insensitive, sensitive, and forward-only asensitive cursors.
  ○ jConnect 5.5 and 6.0.5 support the declaration of insensitive, sensitive, and forward-only asensitive cursors in the same manner as the iAnywhere JDBC driver. However, the underlying implementation of jConnect only supports asensitive cursor semantics.
    For more information about declaring JDBC cursors, see “Requesting SQL Anywhere cursors” on page 55.
- Sybase Open Client supports only asensitive cursors. Also, a severe performance penalty results when using updatable, non-unique cursors.

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 “What are cursors?” on page 29.

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):

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Whitney</td>
</tr>
<tr>
<td>2</td>
<td>Cobb</td>
</tr>
<tr>
<td>3</td>
<td>Chin</td>
</tr>
</tbody>
</table>

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

```sql
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 overview

SQL Anywhere cursors are classified by their sensitivity to changes in the underlying data. In particular, cursor sensitivity is defined in terms of which changes are visible.

- **Insensitive cursors**  The result set is fixed when the cursor is opened. No changes to the underlying data are visible. See “Insensitive cursors” on page 46.
- **Sensitive cursors**  The result set can change after the cursor is opened. All changes to the underlying data are visible. See “Sensitive cursors” on page 46.
- **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 48.
- **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 49.

The differing requirements on cursors place different constraints on execution, and so, performance. See “Cursor sensitivity and performance” on page 50.

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.

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

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Whitney</td>
</tr>
<tr>
<td>105</td>
<td>Cobb</td>
</tr>
<tr>
<td>160</td>
<td>Breault</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch previous row</td>
<td>Returns the original copy of the row (102).</td>
</tr>
<tr>
<td>Fetch the first row (absolute fetch)</td>
<td>Returns the original copy of the row (102).</td>
</tr>
<tr>
<td>Fetch the second row (absolute fetch)</td>
<td>Returns the unchanged row (105).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch previous row</td>
<td>Returns Row Not Found. There is no previous row.</td>
</tr>
<tr>
<td>Fetch the first row (absolute fetch)</td>
<td>Returns row 105.</td>
</tr>
<tr>
<td>Fetch the second row (absolute fetch)</td>
<td>Returns row 160.</td>
</tr>
</tbody>
</table>

• **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.

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

• **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.

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

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Whitney</td>
</tr>
<tr>
<td>105</td>
<td>Cobb</td>
</tr>
<tr>
<td>160</td>
<td>Breault</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch previous row</td>
<td>Returns the original copy of the row (102).</td>
</tr>
<tr>
<td>Fetch the first row (absolute fetch)</td>
<td>Returns the original copy of the row (102).</td>
</tr>
<tr>
<td>Fetch the second row (absolute fetch)</td>
<td>Returns the unchanged row (105).</td>
</tr>
</tbody>
</table>

- **Sensitive cursors**  The membership of the result set has changed so that row 105 is now the first row in the result set:
<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch previous row</td>
<td>Returns Row Not Found. 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.</td>
</tr>
<tr>
<td>Fetch the first row (absolute fetch)</td>
<td>Returns row 105.</td>
</tr>
<tr>
<td>Fetch the second row (absolute fetch)</td>
<td>Returns row 160.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch previous row</td>
<td>Returns Row Not Found. 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.</td>
</tr>
<tr>
<td>Fetch the first row (absolute fetch)</td>
<td>Returns No current row of cursor. 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.</td>
</tr>
<tr>
<td>Fetch the second row (absolute fetch)</td>
<td>Returns row 105.</td>
</tr>
</tbody>
</table>

- **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.

**No warnings or errors in bulk operations mode**

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

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

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

**Standards**

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

**Programming interfaces**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Cursor type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC, ADO/OLE DB</td>
<td>Static</td>
<td>If an updatable static cursor is requested, a value-sensitive cursor is used instead.</td>
</tr>
<tr>
<td>Embedded SQL</td>
<td>INSENSITIVE</td>
<td>Insensitive semantics are only supported by the iAnywhere JDBC driver.</td>
</tr>
<tr>
<td>JDBC</td>
<td>INSENSITIVE</td>
<td></td>
</tr>
<tr>
<td>Open Client</td>
<td>Unsupported</td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Interface</th>
<th>Cursor type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC, ADO/OLE DB</td>
<td>Dynamic</td>
<td></td>
</tr>
<tr>
<td>Embedded SQL</td>
<td>SENSITIVE</td>
<td>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.</td>
</tr>
<tr>
<td>JDBC</td>
<td>SENSITIVE</td>
<td>Sensitive cursors are fully supported by the iAnywhere JDBC driver.</td>
</tr>
</tbody>
</table>

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.

For more information about query optimization and the use of work tables, see “Query optimization and execution” [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 unspecific sensitivity.

Programming interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Cursor type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC, ADO/OLE DB</td>
<td>Unspecified sensitivity</td>
</tr>
<tr>
<td>Embedded SQL</td>
<td>DYNAMIC SCROLL</td>
</tr>
</tbody>
</table>

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

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

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 42.

- 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 No Current Row of Cursor 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 impact performance in some cases.

**Inserting multiple rows**

When inserting multiple rows through a value-sensitive cursor, the new rows appear at the end of the result set. See “Modifying rows through a cursor” on page 36.

**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.

### Prefetching rows

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 [database]” [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 [database]” [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 in cases that are likely to result in improved performance. 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;
   ```

<table>
<thead>
<tr>
<th>ID</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>28</td>
</tr>
<tr>
<td>301</td>
<td>54</td>
</tr>
<tr>
<td>302</td>
<td>75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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 iAnywhere 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 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, on the other hand, 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.

### Requesting 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 SADataAdapter object uses a client-side result set instead of cursors. See “SACommand class” on page 206.

### ADO/OLE DB and ODBC

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

<table>
<thead>
<tr>
<th>ODBC scrollable cursor type</th>
<th>SQL Anywhere cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>Insensitive</td>
</tr>
<tr>
<td>KEYSET-DRIVEN</td>
<td>Value-sensitive</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>Sensitive</td>
</tr>
<tr>
<td>MIXED</td>
<td>Value-sensitive</td>
</tr>
</tbody>
</table>

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 41.
For information about how to request a cursor type in ODBC, see “Choosing ODBC cursor characteristics” on page 492.

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 2.0 and 3.0 specifications support three types of cursors: insensitive, sensitive, and forward-only asensitive. The iAnywhere 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 46.

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:

<table>
<thead>
<tr>
<th>Cursor type</th>
<th>SQL Anywhere cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO SCROLL</td>
<td>Asensitive</td>
</tr>
<tr>
<td>DYNAMIC SCROLL</td>
<td>Asensitive</td>
</tr>
<tr>
<td>SCROLL</td>
<td>Value-sensitive</td>
</tr>
<tr>
<td>INSENSITIVE</td>
<td>Insensitive</td>
</tr>
<tr>
<td>SENSITIVE</td>
<td>Sensitive</td>
</tr>
</tbody>
</table>
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.
Describing result sets

Some applications build SQL statements that cannot be completely specified in the application. In some cases, for example, statements depend on a response from the user 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 565.

- 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].
Controlling 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 “Using transactions and isolation levels” [SQL Anywhere Server - SQL Usage].

Setting autocommit or 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 62.

**To control autocommit mode (ADO.NET)**

- By default, the ADO.NET provider operates in autocommit mode. To use explicit transactions, use the SAConnection.BeginTransaction method. See “Transaction processing” on page 136.

**To 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.
To 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.

To control autocommit mode (JDBC)

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

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

To 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:

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

To 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:

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

To control autocommit mode (PHP)

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

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

See “sasql_set_option” on page 772.

To 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:

```sql
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 “Setting autocommit or manual commit mode” on page 60.
**Autocommit implementation details**

Autocommit mode has slightly different behavior depending on the interface 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, and PHP 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. This behavior is controlled, implicitly in the case of JDBC, by the chained database option.
  
  Embedded SQL, JDBC, and Open Client applications manipulate server-side commit behavior (for example, they set the chained option).

There is a difference between client-side and server-side autocommit in the case of compound statements such as stored procedures or triggers. 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.

---

**Do not mix client-side and server-side implementations**

Do not combine setting of the chained option with setting of the autocommit option in your ADO.NET, ADO/OLE DB, ODBC, or PHP application.

---

**Controlling the isolation level**

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 [database] [compatibility]” [SQL Anywhere Server - Database Administration].

---

**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 [database] [compatibility]” [SQL Anywhere Server - Database Administration].
Three-tier computing and distributed transactions

Contents

Introduction to three-tier computing and distributed transactions ................................ 66
Three-tier computing architecture ................................................................................ 67
Using distributed transactions ..................................................................................... 70
Using EAServer with SQL Anywhere .......................................................................... 72
Introduction to 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 chapter 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 chapter.

**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 chapter 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 OLE DB or ODBC. The ODBC driver and OLE DB provider act as resource manager proxies on the client computer.

- 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. The application developer building the component 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, the application developer does 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 ODBC or OLE DB.

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 DTC service must be running on each computer to operate distributed transactions. You can control DTC services from the Services icon in the Windows Control Panel; the DTC service is named MSDTC.

For more information, see your DTC or EAServer documentation.
Using 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:

<table>
<thead>
<tr>
<th>DTC isolation level</th>
<th>SQL Anywhere isolation level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOLATIONLEVEL_UNSPECIFIED</td>
<td>0</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_CHAOS</td>
<td>0</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_READUNCOMMITTED</td>
<td>0</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_BROWSE</td>
<td>0</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_CURSORSTABILITY</td>
<td>1</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_READONLYCOMMITTED</td>
<td>1</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_REPEATABLEREAD</td>
<td>2</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_SERIALIZABLE</td>
<td>3</td>
</tr>
<tr>
<td>ISOLATIONLEVEL_ISOLATED</td>
<td>3</td>
</tr>
</tbody>
</table>
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. See “-tmf server option” [SQL Anywhere Server - Database Administration].

- `-tmt` If re-enlistment is not achieved before the specified time, the outstanding operations are rolled back and recovery continues. See “-tmt server option” [SQL Anywhere Server - Database Administration].
Using EAServer with SQL Anywhere

This section provides an overview of the actions you need to take in EAServer 3.0 or later to work with SQL Anywhere. For more details, see the EAServer documentation.

Configuring EAServer

All components installed in a Sybase EAServer system share the same transaction coordinator.

EAServer 3.0 and later offer a choice of transaction coordinators. You must use DTC as the transaction coordinator if you are including SQL Anywhere in the transactions. This section describes how to configure EAServer 3.0 to use DTC as its transaction coordinator.

The component server in EAServer is named Jaguar.

To configure an EAServer to use the Microsoft DTC transaction model

1. Ensure that your Jaguar server is running.
   
   On Windows, the Jaguar server commonly runs as a service. To manually start the installed Jaguar server that is included with EAServer 3.0, choose Start » Programs » Sybase » EAServer » EAServer.

2. Start Jaguar Manager.
   
   From the Windows desktop, choose Start » Programs » Sybase » EAServer » Jaguar Manager.

3. Connect to the Jaguar server from Jaguar Manager.
   
   From Sybase Central, choose Tools » Connect » Jaguar Manager. In the connection window, enter jagadmin as the User Name, leave the Password field blank, and enter a Host Name of localhost. Click OK to connect.

4. Set the transaction model for the Jaguar server.
   
   In the left pane, open the Servers folder. In the right pane, right-click the server you want to configure, and choose Server Properties. Click the Transactions tab, and choose Microsoft DTC as the transaction model. Click OK to complete the operation.

Setting the component transaction attribute

In EAServer, you can implement a component that carries out operations on more than one database. You assign a transaction attribute to this component that defines how it participates in transactions. The transaction attribute can have the following values:

- **Not Supported** The component's methods never execute as part of a transaction. If the component is activated by another component that is executing within a transaction, the new instance's work is performed outside the existing transaction. This is the default.

- **Supports Transaction** The component can execute in the context of a transaction, but a connection is not required to execute the component's methods. If the component is instantiated directly by a base
client, EAServer does not begin a transaction. If component A is instantiated by component B, and component B is executing within a transaction, component A executes in the same transaction.

- **Requires Transaction** The component always executes in a transaction. When the component is instantiated directly by a base client, a new transaction begins. If component A is activated by component B, and B is executing within a transaction, then A executes within the same transaction; if B is not executing in a transaction, then A executes in a new transaction.

- **Requires New Transaction** Whenever the component is instantiated, a new transaction begins. If component A is activated by component B, and B is executing within a transaction, then A begins a new transaction that is unaffected by the outcome of B's transaction; if B is not executing in a transaction, then A executes in a new transaction.

For example, in the Sybase Virtual University sample application, included with EAServer as the SVU package, the SVUEnrollment component enroll method carries out two separate operations (reserves a seat in a course, bills the student for the course). These two operations need to be treated as a single transaction.

Microsoft Transaction Server provides the same set of attribute values.

**To set the transaction attribute of a component**

1. In Jaguar Manager, locate the component.

   To find the SVUEnrollment component in the Jaguar sample application, connect to the Jaguar server, open the Packages folder, and open the SVU package. The components in the package are listed in the right pane.

2. Set the transaction attribute for the component.

   Right-click the component, and choose Component Properties. Click the Transaction tab, and choose the transaction attribute value from the list. Click OK to complete the operation.

   The SVUEnrollment component is already marked as Requires Transaction.

Once the component transaction attribute is set, you can perform SQL Anywhere database operations from that component, and be assured of transaction processing at the level you have specified.
Java in the database

This section provides an introduction to Java and Java in the database.

Java support in SQL Anywhere ................................................................. 77
Introduction to Java support

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:

- You can 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.

<table>
<thead>
<tr>
<th>Title</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Java support in SQL Anywhere” on page 77 (this chapter)</td>
<td>Java concepts and how to apply them in SQL Anywhere.</td>
</tr>
<tr>
<td>“Creating a Java class for use with SQL Anywhere” on page 89</td>
<td>Practical steps for using Java in the database.</td>
</tr>
<tr>
<td>“SQL Anywhere JDBC driver” on page 501</td>
<td>Accessing data from Java classes, including distributed computing.</td>
</tr>
</tbody>
</table>

The following table is a guide to which parts of the Java documentation apply to you, depending on your interests and background.

<table>
<thead>
<tr>
<th>If you ...</th>
<th>Consider reading ...</th>
</tr>
</thead>
</table>
| Are a Java developer who wants to just get started. | “The runtime environment for Java in the database” on page 85  
<p>|                                  | “Creating a Java class for use with SQL Anywhere” on page 89 |</p>
<table>
<thead>
<tr>
<th>If you ...</th>
<th>Consider reading ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Want to know the key features of Java in the database.</td>
<td>“Java in the database Q &amp; A” on page 80</td>
</tr>
<tr>
<td>Want to find out how to access data from Java.</td>
<td>“SQL Anywhere JDBC driver” on page 501</td>
</tr>
</tbody>
</table>
Java in the database Q & A

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 server** An external Java Virtual Machine (VM) runs Java code in the database server.
- **You can access data from Java** An internal JDBC driver 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.

For more information, see “Installing Java classes into a database” on page 98.

How does Java get executed in a database?

SQL Anywhere uses a Java Virtual Machine (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.
- Java code is interpreted, so no operations get executed without being acceptable to the Java VM.

On what platforms is Java in the database supported?

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 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.

How do I access Java from SQL?

You can treat Java methods as stored procedures, which can be called from SQL. You must create a stored procedure that runs your method. For example:

```
CREATE PROCEDURE javaproc()
EXTERNAL NAME 'JDBCExample.MyMethod ()V'
LANGUAGE JAVA;
```

For more information, see “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference].

For example, the SQL function PI( * ) returns the value for pi. The Java API class java.lang.Math has a parallel field named PI returning the same value. But java.lang.Math also has a field named E that returns the base of the natural logarithms, and a method that computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.

Other members of the Java API offer even more specialized functionality. For example, java.util.Stack generates a last-in, first-out queue that can store ordered lists; java.util.HashTable maps values to keys; and java.util.StringTokenizer breaks a string of characters into individual word units.
How can I use my own Java classes in databases?

You can install your own Java classes into a database. For example, you could design, write in Java, and compile with a Java compiler, a user-created Employees class or Package class.

User-created Java classes can contain both information about the subject and some computational logic. Once installed in a database, SQL Anywhere lets you use these classes in all parts and operations of the database and execute their functionality (in the form of class or instance methods) as easily as calling a stored procedure.

Java classes and stored procedures are different

Java classes are different from stored procedures. Whereas stored procedures are written in SQL, Java classes provide a more powerful language, and can be called from client applications as easily and in the same way as stored procedures.

For more information, see “Installing Java classes into a database” on page 98.

Can I access data using Java?

The JDBC interface is an industry standard, designed specifically to access database systems. The JDBC classes are designed to connect to a database, request data using SQL statements, and return result sets that can be processed in the client application.

Normally, client applications use JDBC classes, and the database system vendor supplies a JDBC driver that allows the JDBC classes to establish a connection.

You can connect to SQL Anywhere from a client application via JDBC, using jConnect, or using the iAnywhere JDBC driver. SQL Anywhere also provides an internal JDBC driver that permits Java classes installed in a database to use JDBC classes that execute SQL statements. See “SQL Anywhere JDBC driver” on page 501.

Can I move classes from client to server?

You can create Java classes that can be moved between levels of an enterprise application. The same Java class can be integrated into either the client application, a middle tier, or the database—wherever is most appropriate.

You can move a class containing business logic to any level of the enterprise system, including the database server, allowing you complete flexibility to make the most appropriate use of resources. It also enables enterprise customers to develop their applications using a single programming language in a multi-tier architecture with unparalleled flexibility.

What can I not do with Java in the database?

SQL Anywhere is a runtime environment for Java classes, not a Java development environment.

You cannot perform the following tasks in the database:
● Edit class source files (*.java files).
● Compile Java class source files (*.java files).
● Execute unsupported Java APIs, such as applet and visual classes.
● Execute Java methods that require the execution of native methods. All user classes installed into the database must be 100% Java.

The Java classes used in SQL Anywhere must be written and compiled using a Java application development tool, and then installed into a database for use.
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 runtime environment for Java in the database

This section describes the SQL Anywhere runtime environment for Java, and how it differs from a standard Java runtime environment.

The runtime Java classes

The runtime Java classes are the low-level classes that are made available to a database when it is created or Java-enabled. These classes include a subset of the Java API. These classes are part of the Sun Java Development Kit.

The runtime classes provide basic functionality on which to build applications. The runtime classes are always available to classes in the database.

You can incorporate the runtime Java classes in your own user-created classes: either inheriting their functionality or using it within a calculation or operation in a method.

Examples

Some Java API classes included in the runtime Java classes include:

- **Primitive Java data types**  All primitive (native) data types in Java have a corresponding class. In addition to being able to create objects of these types, the classes have additional, often useful, functionality.
  
  The Java int data type has a corresponding class in java.lang.Integer.

- **The utility package**  The package java.util.* contains several classes with functionality that is unavailable in the SQL Anywhere SQL functions.
  
  Some of the classes include:
  
  - Nullable
  - String
  - Boolean
  - Integer
  - Float
  - Double
  - Long
  
- **JDBC for SQL operations**  The package java.SQL.* contains the classes needed by Java objects to extract data from the database using SQL statements.

Unlike user-defined classes, the runtime classes are not stored in the database. Instead, they are stored where the Sun JRE is installed.
Java is case sensitive

Java syntax works as you would expect it to, and SQL syntax is unaltered by the presence of Java classes. This is true even if the same SQL statement contains both Java and SQL syntax. It is a simple statement, but with far-reaching implications.

Java is case sensitive. The Java class FindOut is a completely different class from the class Findout. For keywords and identifiers SQL is case insensitive.

Java case sensitivity is preserved even when embedded in a SQL statement that is case insensitive. The Java parts of the statement must be case sensitive, even though the parts previous to and following the Java syntax can be in either upper or lowercase.

For example, the following SQL statement executes successfully because the case of Java objects, classes, and operators is respected even though there is variation in the case of the remaining SQL parts of the statement.

```sql
SELECT java.lang.Math.random();
```

Strings in Java and SQL

A set of double quotes identifies string literals in Java, as in the following Java code fragment:

```java
String str = "This is a string";
```

In SQL, however, single quotes mark strings, and double quotes indicate an identifier, as illustrated by the following SQL statement:

```sql
INSERT INTO TABLE DBA.t1 VALUES( 'Hello' );
```

You should always use the double quote in Java source code, and single quotes in SQL statements.

The following Java code fragment is valid, if used within a Java class.

```java
String str = new java.lang.String( "Brand new object" );
```

Printing to the command line

Printing to the standard output is a quick way of checking variable values and execution results at various points of code execution. When the method in the second line of the following Java code fragment is encountered, the string argument it accepts prints out to standard output.

```java
String str = "Hello world";
System.out.println( str );
```

In SQL Anywhere, standard output is the database server messages window, so the string appears there. Executing the above Java code within the database is the equivalent of the following SQL statement.

```sql
MESSAGE 'Hello world';
```
**Using the main method**

When a class contains a main method matching the following declaration, most Java run time environments, such as the Sun Java interpreter, execute it automatically. Normally, this static method executes only if it is the class being invoked by the Java interpreter.

```java
public static void main( String args[] ) { }
```

You are always guaranteed this method will be called first when the Sun Java runtime system starts.

In SQL Anywhere, the Java runtime system is always available. The functionality of objects and methods can be tested in an ad hoc, dynamic manner using SQL statements. This provides a flexible method for testing Java class functionality.

**Persistence**

Once a Java class has been added to a database, it remains there until you explicitly remove it with a REMOVE JAVA statement.

Variables in Java classes, like SQL variables, persist only for the duration of the connection.

For more information about removing classes, see “REMOVE JAVA statement” [SQL Anywhere Server - SQL Reference].

**Java escape characters in SQL statements**

In Java code, you can use escape characters to insert certain special characters into strings. Consider the following code, which inserts a new line and tab in front of a sentence containing an apostrophe.

```java
String str = "\n\t\This is an object\'s string literal";
```

SQL Anywhere permits the use of Java escape characters only when being used by Java classes. From within SQL, however, you must follow the rules that apply to strings in SQL.

For example, to pass a string value to a field using a SQL statement, you could use the following statement (which includes SQL escape characters), but the Java escape characters could not be used.

```sql
SET obj.str = '\nThis is the object\'s string field';
```

For more information about SQL string handling rules, see “Strings” [SQL Anywhere Server - SQL Reference].

**Using import statements**

It is common in a Java class declaration to include an import statement to access classes in another package. You can reference imported classes using unqualified class names.

For example, you can reference the Stack class of the java.util package in two ways:
• explicitly using the name java.util.Stack
• using the name Stack, and including the following import statement:

    import java.util.*;

**Classes further up in the hierarchy must also be installed**

A class referenced by another class, either explicitly with a fully qualified name or implicitly using an import statement, must also be installed in the database.

The import statement works as intended within compiled classes. However, within the SQL Anywhere runtime environment, no equivalent to the import statement exists. All class names used in stored procedures must be fully qualified. For example, to create a variable of type String, you would reference the class using the fully qualified name: java.lang.String.

**Public fields**

It is a common practice in object-oriented programming to define class fields as private and make their values available only through public methods.

Many of the examples used in this documentation render fields public to make examples more compact and easier to read. Using public fields in SQL Anywhere also offers a performance advantage over accessing public methods.

The general convention followed in this documentation is that a user-created Java class designed for use in SQL Anywhere exposes its main values in its fields. Methods contain computational automation and logic that may act on these fields.
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 samples-dir\SQLAnywhere\JavaInvoice.

The first step to using Java in the database is to write the Java code and compile it. This is done outside the database.

**To create and compile the class**

1. Create the sample Java class source file.

   For your convenience, the sample code is included here. You can paste the following code into Invoice.java or obtain the file from samples-dir\SQLAnywhere\JavaInvoice.

   ```java
   import java.io.*;
   public class Invoice {
   public static String lineItem1Description;
   public static double lineItem1Cost;
   public static String lineItem2Description;
   public static double lineItem2Cost;
   public static double totalSum() {
      double runningsum;
      double taxfactor = 1 + Invoice.rateOfTaxation();
      runningsum = lineItem1Cost + lineItem2Cost;
      runningsum = runningsum * taxfactor;
      return runningsum;
   }
   public static double rateOfTaxation() {
      double rate;
      rate = .15;
      return rate;
   }
   public static void init(String item1desc, double item1cost,
               String item2desc, double item2cost) {
      lineItem1Description = item1desc;
      lineItem1Cost = item1cost;
      lineItem2Description = item2desc;
      lineItem2Cost = item2cost;
   }
   public static String getLineItem1Description() {
   }
   ```
{    return lineItem1Description; }

public static double getLineItem1Cost()
{    return lineItem1Cost; }

public static String getLineItem2Description()
{    return lineItem2Description; }

public static double getLineItem2Cost()
{    return lineItem2Cost; }

public static boolean testOut( int[] param )
{    param[0] = 123;
    return true;
}

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();
}

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.
Choosing a Java VM

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:\jdk1.5.0_06\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 information is not in the path, return an error.

See “ALTER EXTERNAL ENVIRONMENT statement” [SQL Anywhere Server - SQL Reference].

**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.

### To 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:\jdk1.5.0_06\jre\bin`).

You can also use the ALTER EXTERNAL ENVIRONMENT to specify the database user whose connection can be used for installing classes and performing other Java-related administrative tasks.

```
ALTER EXTERNAL ENVIRONMENT JAVA
USER user_name
```

For more information, see “ALTER EXTERNAL ENVIRONMENT statement” [SQL Anywhere Server - SQL Reference].

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';
```

For more information, see “java_vm_options option [database]” [SQL Anywhere Server - Database Administration].
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.
Install 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.

To 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.
5. Click Browse and browse to the location of Invoice.class.
6. Click Finish.

To 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:

   ```sql
   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.

For more information about installing classes, and for information about updating an installed class, see “Installing Java classes into a database” on page 98.
Using the CLASSPATH variable

The Sun Java runtime environment and the Sun 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 install the class.
Accessing 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.

**To call a Java method using Interactive SQL**

1. Create the following SQL stored procedure to call the Invoice.main method in the sample class:

   ```sql
   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.
   
   For more information about the syntax of this statement, see “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference].

2. Call the stored procedure to call the Java method:

   ```sql
   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.
Accessing fields and methods of the Java object

Here are more examples of how to call Java methods, pass arguments, and return values.

To 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:

```sql
-- 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;
-- Invoice.rateOfTaxation take no arguments ()
-- and returns a double (D)
CREATE FUNCTION rateOfTaxation()
RETURNS DOUBLE
EXTERNAL NAME
   'Invoice.rateOfTaxation()D'
LANGUAGE JAVA;
-- Invoice.getLineItem1Description take no arguments ()
-- and returns a string (Ljava/lang/String;)
CREATE FUNCTION getLineItem1Description()
RETURNS CHAR(50)
EXTERNAL NAME
   'Invoice.getLineItem1Description()Ljava/lang/String;' LANGUAGE JAVA;
-- Invoice.getLineItem1Cost take no arguments ()
-- and returns a double (D)
CREATE FUNCTION getLineItem1Cost()
RETURNS DOUBLE
EXTERNAL NAME
   'Invoice.getLineItem1Cost()D'
LANGUAGE JAVA;
-- Invoice.getLineItem2Description take no arguments ()
-- and returns a string (Ljava/lang/String;)
CREATE FUNCTION getLineItem2Description()
RETURNS CHAR(50)
EXTERNAL NAME
   'Invoice.getLineItem2Description()Ljava/lang/String;' LANGUAGE JAVA;
-- Invoice.getLineItem2Cost take no arguments ()
-- and returns a double (D)
CREATE FUNCTION getLineItem2Cost()
RETURNS DOUBLE
EXTERNAL NAME
   'Invoice.getLineItem2Cost()D'
```
The descriptors for arguments to and return values from Java methods have the following meanings:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>byte</td>
</tr>
<tr>
<td>C</td>
<td>char</td>
</tr>
<tr>
<td>D</td>
<td>double</td>
</tr>
<tr>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>I</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>long</td>
</tr>
<tr>
<td>L class-name;</td>
<td>An instance of the class class-name. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String.</td>
</tr>
<tr>
<td>S</td>
<td>short</td>
</tr>
<tr>
<td>V</td>
<td>void</td>
</tr>
<tr>
<td>Z</td>
<td>Boolean</td>
</tr>
<tr>
<td>[</td>
<td>Use one for each dimension of an array.</td>
</tr>
</tbody>
</table>

For more information about the syntax of these statements, see “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference] and “CREATE FUNCTION statement (web services)” [SQL Anywhere Server - SQL Reference].

2. Call the stored procedure that is acting as a wrapper to call the Java method:

```java
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:

<table>
<thead>
<tr>
<th>Item1</th>
<th>Item1Cost</th>
<th>Item2</th>
<th>Item2Cost</th>
<th>TaxRate</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>10</td>
<td>Jacket</td>
<td>25</td>
<td>0.15</td>
<td>40.25</td>
</tr>
</tbody>
</table>
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 Sun 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:

**To create a class**

1. **Define your class.**
   
   Write the Java code that defines your class. If you are using the Sun Java SDK then you can use a text editor. If you are using a development tool, the development tool provides instructions.

   **Use only supported classes**
   
   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 Sun 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.

You require DBA authority to install a class.
To install a class (Sybase Central)

1. Connect to a database as a DBA user.
2. Open the External Environments folder.
3. Under this folder, open the Java folder.
4. Right-click the right pane and choose New » Java Class.
5. Follow the instructions in the wizard.

To install a class (SQL)

1. Connect to the database as a DBA user.
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.

For more information, see “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`.

For more information, see “INSTALL JAVA statement” [SQL Anywhere Server - SQL Reference].

To install a JAR (Sybase Central)

1. Connect to the database as a DBA user.
2. Open the External Environments folder.
3. Under this folder, open the Java folder.
4. Right-click the right pane and choose **New » JAR File**.
5. Follow the instructions in the wizard.

**To install a JAR (SQL)**

1. Connect to a database as a DBA user.
2. Execute the following statement:

   ```sql
   INSTALL JAVA NEW
   JAR 'jarname'
   FROM FILE 'path\JarName.jar';
   ```

**Updating 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.

**To update a class or JAR (Sybase Central)**

1. Connect to the database as a DBA user.
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. Select the class or JAR file and the choose **File » Update**.
6. In the **Update** window, specify the name and location of the class or JAR file to be updated. You can click **Browse** to search for it.

**Tips**

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.
To update a class or JAR (SQL)

1. Connect to a database as a DBA user.
2. Execute the following statement:

   ```sql
   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 “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 samples-dir\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
```

To call the main method of a class from SQL

1. Declare the method with an array of strings as an argument:

   ```java
   public static void main( java.lang.String args[] )
   {
   ...
   }
   ```

2. Create a stored procedure that wraps this method.

   ```sql
   CREATE PROCEDURE JavaMain( in arg char(50) )
   EXTERNAL NAME 'JavaClass.main([Ljava/lang/String;)V'
   LANGUAGE JAVA;
   ```

   For more information, see “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference].

3. Invoke the main method using the CALL statement.

   ```sql
   call JavaMain( 'Hello world' );
   ```

   Due to the limitations of the SQL language, only a single string can be passed.

Using 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.
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.

To 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 (web services)” [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.

```java
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:

```sql
CREATE PROCEDURE result_set()  
DYNAMIC RESULT SETS 1
```
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.

For more information about Java method signatures, see “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference].

For more information about returning result sets, see “Returning result sets” on page 523.

Returning values 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:

```java
public class Invoice
{
    public static boolean testOut( int[] param )
    {
        param[0] = 123;
        return true;
    }
}
```

The following procedure uses the testOut method:

```sql
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.

```sql
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 (web services)” [SQL Anywhere Server - SQL Reference].
Security management for Java

Java provides security managers than 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;
SQL Anywhere Data Access APIs

This section describes the programming interfaces for SQL Anywhere.
SQL Anywhere .NET Data Provider

Contents

SQL Anywhere .NET Data Provider features ................................................................. 110
Running the sample projects ..................................................................................... 111
Using the .NET Data Provider in a Visual Studio project ........................................... 112
Connecting to a database .......................................................................................... 114
Accessing and manipulating data .............................................................................. 117
Using stored procedures ........................................................................................... 134
Transaction processing ............................................................................................. 136
Error handling and the SQL Anywhere .NET Data Provider ...................................... 138
Deploying the SQL Anywhere .NET Data Provider ................................................... 139
Tracing support .......................................................................................................... 141
SQL Anywhere .NET Data Provider features

SQL Anywhere supports the Microsoft .NET Framework version 2.0 or later 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 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.

- **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.
Running the sample projects

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

- **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**.

- **SimpleXML**  A .NET Framework sample project for Windows that demonstrates how to obtain XML data from SQL Anywhere via ADO.NET.

- **TableViewer**  A .NET Framework sample project for Windows that allows you to enter and execute SQL statements.

For tutorials explaining the sample projects, see “Tutorial: Using the SQL Anywhere .NET Data Provider” on page 145.

**Note**
If you installed SQL Anywhere in a location other than the default installation directory (\Program Files\SQL Anywhere 11), you may receive an error referencing the Data Provider DLL when you load the sample projects. If this happens, add a new reference to `iAnywhere.Data.SQLAnywhere.dll`. There is one version of the Data Provider that supports the .NET Framework 2.0 and later versions. The Data Provider for Windows is located in `install-dir\Assembly\v2\iAnywhere.Data.SQLAnywhere.dll`. The Data Provider for Windows Mobile is located in `install-dir\ce\Assembly`.

For instructions on adding a reference to the DLL, see “Adding a reference to the Data Provider DLL in your project” on page 112.
Using the .NET Data Provider in a Visual Studio project

The SQL Anywhere .NET Data Provider can be used to develop applications with Visual Studio 2005 or later versions. To use the SQL Anywhere .NET Data Provider, you must include two items in your Visual Studio project:

- a reference to the SQL Anywhere .NET Data Provider DLL
- a line in your source code referencing the SQL Anywhere .NET Data Provider classes

These steps are explained below.

For information about installing and registering the SQL Anywhere .NET Data Provider, see “Deploying the SQL Anywhere .NET Data Provider” on page 139.

Adding a reference to the Data Provider DLL in your project

Adding a reference tells Visual Studio which DLL to include to find the code for the SQL Anywhere .NET Data Provider.

To add a reference to the SQL Anywhere .NET Data Provider in a Visual Studio project

1. Start Visual Studio and open your project.
2. In the Solution Explorer window, right-click References and choose Add Reference.
3. On the .NET tab, click Browse to locate iAnywhere.Data.SQLAnywhere.dll. Note that there are separate versions of the DLL for each of Windows and Windows Mobile platforms.
   - For the Windows SQL Anywhere .NET Data Provider, the default location is install-dir\Assembly\v2.
   - For the Windows Mobile SQL Anywhere .NET Data Provider, the default location is install-dir\ce\Assembly\v2.
4. Select the DLL and then click Open.
   For a complete list of installed DLLs, see “SQL Anywhere .NET Data Provider required files” on page 139.
5. You can verify that the DLL is added to your project. Open the Add Reference window and then click the .NET tab. iAnywhere.Data.SQLAnywhere.dll appears in the Selected Components list. Click OK to close the window.

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

Using the Data Provider classes in your source code

To assist with the use of the SQL Anywhere .NET Data Provider namespace and the types defined in this namespace, you should add a directive to your source code.
To use the Data Provider namespace in your code

1. Start Visual Studio and open your project.
2. 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
```

This directive is not required, however, it allows you to use short forms for the SQL Anywhere .NET classes. For example:

```
SAConnection conn = new SAConnection()
```

Without this directive, you can still use the following:

```
iAnywhere.Data.SQLAnywhere.SAConnection
conn = new iAnywhere.Data.SQLAnywhere.SAConnection()
```
Connecting to a database

Before you can perform any operations on the data, your application must connect to the database. This section describes how to write code to connect to a SQL Anywhere database.

For more information, see “SAConnectionStringBuilder class” on page 269 and “ConnectionName property” on page 278.

To connect to a SQL Anywhere database

1. Allocate an SAConnection object.

   The following code creates an SAConnection object named `conn`:

   ```csharp
   SAConnection conn = new SAConnection(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:

   ```csharp
   private SAConnection _conn;
   ```

   For more information, see the sample code in `samples-dir\SQLAnywhere\ADO.NET\TableViewer` and “Understanding the Table Viewer sample project” on page 152.

2. Specify the connection string used to connect to the database.

   For example:

   ```plaintext
   "Data Source=SQL Anywhere 11 Demo"
   ```

   For a complete list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

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

3. Open a connection to the database.

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

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


   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:

   ```csharp
   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 = "Data Source=SQL Anywhere 11 Demo";
_conn.Open();

5. Close the connection to the database. Connections to the database stay open until they are explicitly closed using the conn.Close() method.

Visual Basic connection example

The following Visual Basic code 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 11 Demo"
        myConn.Open()
        myConn.Close()
    End Sub

Connection pooling

The SQL Anywhere .NET Data Provider supports 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 11 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.

See also

- “ConnectionName property” on page 278
- “AutoStop connection parameter [ASTOP]” [SQL Anywhere Server - Database Administration]

Checking the 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 SAConnection class has a state property that checks the state of the connection. Possible state values are Open and Closed.

The following code checks whether the Connection 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.

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

For more information, see “State property” on page 254.
Accessing and manipulating data

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

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

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

  Within an SACommand object, an SDataReader is used to return read-only result sets from a query or stored procedure. The SDataReader 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 SACommand 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 “SACommand class” on page 206 and “SDataReader class” on page 310.

- **SDataAdapter object**  The SDataAdapter 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 SDataAdapter 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 125.

  For more information about the SDataAdapter object, see “SDataAdapter class” on page 299.

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.

Getting 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.

For more information, see “ExecuteReader methods” on page 226.

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.

For more information, see “ExecuteScalar method” on page 227.

When using the SACommand object, you can use the SADataReader 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.

The following instructions use the Simple code sample included with the .NET Data Provider.

For more information about the Simple code sample, see “Understanding the Simple sample project” on page 148.

To issue a SQL query that returns a complete result set

1. Declare and initialize a Connection object.

   ```csharp
   SAConnection conn = new SAConnection(
     "Data Source=SQL Anywhere 11 Demo" );
   ```

2. Open the connection.

   ```csharp
   try {
     conn.Open();
   }
   ```

3. Add a Command object to define and execute a SQL statement.

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

   If you are calling a stored procedure, you must specify the parameters for the stored procedure.

   For more information, see “Using stored procedures” on page 134 and “SAParameter class” on page 382.

4. Call the ExecuteReader method to return the DataReader object.

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

5. Display the results.

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

6. Close the DataReader and Connection objects.

   ```csharp
   reader.Close();
   conn.Close();
   ```
To issue a SQL query that returns only one value

1. Declare and initialize an SAConnection object.

   ```java
   SAConnection conn = new SAConnection(
       "Data Source=SQL Anywhere 11 Demo" );
   ```

2. Open the connection.

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

3. Add an SACommand object to define and execute a SQL statement.

   ```java
   SACommand cmd = new SACommand(
       "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'",
       conn );
   ```

   If you are calling a stored procedure, you must specify the parameters for the stored procedure.

   For more information, see “Using stored procedures” on page 134.

4. Call the ExecuteScalar method to return the object containing the value.

   ```java
   int count = (int) cmd.ExecuteScalar();
   ```

5. Close the SAConnection object.

   ```java
   conn.Close();
   ```

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

For more information, see “SDataReader class” on page 310.

**Visual Basic DataReader example**

The following Visual Basic code opens a connection to the SQL Anywhere sample database and uses the DataReader to return the last name of the first five employees in the result set:

```vbnet
Dim myConn As New .SAConnection()
Dim myCmd As New .SACommand("
   SELECT Surname FROM Employees",
   myConn)
Dim myReader As SDataReader
Dim counter As Integer
myConn.ConnectionString = 
   "Data Source=SQL Anywhere 11 Demo"
myConn.Open()
myReader = myCmd.ExecuteReader()
counter = 0
Do While (myReader.Read())
   MsgBox(myReader.GetString(0))
   counter = counter + 1
   If counter >= 5 Then Exit Do
Loop
myConn.Close()
```
Inserting, updating, and deleting rows using the SACommand object

To perform an insert, update, or delete with the SACommand object, use the ExecuteNonQuery function. The ExecuteNonQuery function issues a query (SQL statement or stored procedure) that does not return a result set. See “ExecuteNonQuery method” on page 225.

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 SACommand object.

For information about obtaining primary key values for autoincrement primary keys, see “Obtaining primary key values” on page 129.

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 .NET Data Provider operates in autocommit mode and any changes that you make are applied immediately. See “Transaction processing” on page 136.

To issue a statement that inserts a row

1. Declare and initialize an SAConnection object.

   ```csharp
   SAConnection conn = new SAConnection(c_connStr );
   conn.Open();
   ```

2. Open the connection.

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

3. Add an SACommand object to define and execute an INSERT statement.

   You can use an INSERT, UPDATE, or DELETE statement with the ExecuteNonQuery method.

   ```csharp
   SACommand insertCmd = new SACommand(
       "INSERT INTO Departments( DepartmentID, DepartmentName )
       VALUES( ?, ? )", conn );
   ```

   If you are calling a stored procedure, you must specify the parameters for the stored procedure.

   For more information, see “Using stored procedures” on page 134 and “SAParameter class” on page 382.

4. Set the parameters for the SACommand object.

   The following code defines parameters for the DepartmentID and DepartmentName columns respectively.

   ```csharp
   SAParameter parm = new SAParameter();
   parm.SADbType = SADbType.Integer;
   insertCmd.Parameters.Add( parm );
   parm = new SAParameter();
   parm.SADbType = SADbType.Char;
   insertCmd.Parameters.Add( parm );
   ```

5. Insert the new values and call the ExecuteNonQuery method to apply the changes to the database.

   ```csharp
   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();

6. Display the results and bind them to the grid on the screen.

   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(10, 10);
   dataGrid.Size = new Size(275, 200);
   dataGrid.CaptionText = "iAnywhere SACommand Example";
   this.Controls.Add(dataGrid);

   dataGrid.DataSource = dr;
   dataGrid.Show();

7. Close the SADataReader and SAConnection objects.

   dr.Close();
   conn.Close();

To issue a statement that updates a row

1. Declare and initialize an SAConnection object.

   SACommand conn = new SACommand(  
       c_connStr );

2. Open the connection.

   conn.Open();

3. Add an SACommand object to define and execute an UPDATE statement.

   You can use an INSERT, UPDATE, or DELETE statement with the ExecuteNonQuery method.

   SACommand updateCmd = new SACommand(  
       "UPDATE Departments SET DepartmentName = 'Engineering'  
       WHERE DepartmentID=100", conn );

   If you are calling a stored procedure, you must specify the parameters for the stored procedure.

   For more information, see “Using stored procedures” on page 134 and “SAPassword  
   class” on page 382.

4. Call the ExecuteNonQuery method to apply the changes to the database.

   int recordsAffected = updateCmd.ExecuteNonQuery();

5. Display the results and bind them to the grid on the screen.

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

6. Close the SADataReader and SAConnection objects.
dr.Close();
conn.Close();

To issue a statement that deletes a row

1. Declare and initialize an SAConnection object.

   SAConnection conn = new SAConnection(c_connStr);

2. Open the connection.

   conn.Open();

3. Create an SACommand object to define and execute a DELETE statement.

   You can use an INSERT, UPDATE, or DELETE statement with the ExecuteNonQuery method.

   SACmd deleteCmd = new SACmd(
   "DELETE FROM Departments WHERE ( DepartmentID > 500 )", conn);

   If you are calling a stored procedure, you must specify the parameters for the stored procedure.

   For more information, see “Using stored procedures” on page 134 and “SAParameter class” on page 382.

4. Call the ExecuteNonQuery method to apply the changes to the database.

   int recordsAffected = deleteCmd.ExecuteNonQuery();

5. Close the SAConnection object.

   conn.Close();

Obtaining DataReader schema information

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

If you are using the SADataReader, you can use the GetSchemaTable method to obtain information about the 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.

For more information about the GetSchemaTable method, see “GetSchemaTable method” on page 331.

To obtain information about a result set using the GetSchemaTable method

1. Declare and initialize a connection object.

   SAConnection conn = new SAConnection(c_connStr);

2. Open the connection.

   conn.Open();

3. Create an SACommand object with the SELECT statement you want to use. The schema is returned for the result set of this query.
SACMDcmd = new SACMDcmd(
"SELECT * FROM Employees", conn);

4. Create an SADatReader object and execute the Command object you created.

SADatReader dr = cmd.ExecuteReader();

5. Fill the DataTable with the schema from the data source.

DataTable schema = dr.GetSchemaTable();

6. Close the SADatReader and SAConn object.

dr.Close();
conn.Close();

7. Bind the DataTable to the grid on the screen.

dataGrid.DataSource = schema;

**Using the SADatAdapter object to access and manipulate data**

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

**Getting data using the SADatAdapter object**

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

Using the SADatAdapter, you can pass any string (SQL statement or stored procedure) that returns a result set. When you use the SADatAdapter, 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 SADatAdapter allows you to make changes to the DataSet. Once your changes are complete, you must reconnect to the database to apply the changes.

You can use the SADatAdapter 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 DataSet 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 DataSet are applied to the database in the event that another user changes the data you are modifying before your changes are applied to the database.

For more information about the SADatAdapter, see “SADatAdapter class” on page 299.
**SADataAdapter example**

The following example shows how to fill a DataSet using the SADataAdapter.

**To retrieve data using the SADataAdapter object**

1. Connect to the database.
2. Create a new DataSet. In this case, the DataSet is called Results.
   ```csharp
   DataSet ds = new DataSet();
   ```
3. Create a new SADataAdapter object to execute a SQL statement and fill the DataSet.
   ```csharp
   SADataAdapter da = new SADataAdapter(txtSQLStatement.Text, _conn);
   da.Fill(ds, "Results")
   ```
4. Bind the DataSet to the grid on the screen.
   ```csharp
dgResults.DataSource = ds.Tables["Results"]
   ```

**Inserting, updating, and deleting rows using the SADataAdapter object**

The SADataAdapter retrieves the result set into a DataSet. A DataSet is a collection of tables 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 the DataSet and to update the database with changes made to the DataSet. However, once the DataSet is filled, you can modify the DataSet 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.

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 SADataAdapter 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 in the event that another user changes the data you are modifying before your changes are applied to the database.
Resolving conflicts when using the SADataAdapter

When you use the SADataAdapter, 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 “Obtaining primary key values” on page 129.

- **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 SADataAdapter 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, SADataReader, 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 “Inserting, updating, and deleting rows using the SACommand object” on page 120.

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.

To insert rows into a table using the SADataAdapter

1. Declare and initialize an SAConnection object.

   ```csharp
   SAConnection conn = new SAConnection(c_connStr);
   ```

2. Open the connection.

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

3. Create a new SDataAdapter object.

   ```csharp
   SDataAdapter adapter = new SDataAdapter();
   ```

4. Create the necessary SACommand objects and define any necessary parameters.

The following code creates a SELECT and an INSERT statement and defines the parameters for the INSERT statement.

```csharp
adapter.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn);
adapter.InsertCommand = new SACommand(
    "INSERT INTO Departments( DepartmentID, DepartmentName )
    VALUES( ?, ? )", conn);
adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.None;
SAParameter parm = new SAParameter();
parm.SADBType = SADBType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add(parm);
parm = new SAParameter();
parm.SADBType = SADBType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add(parm);

5. Fill the DataTable with the results of the SELECT statement.

```csharp
DataTable dataTable = new DataTable( "Departments" );
int rowCount = adapter.Fill( dataTable );
```

6. Insert the new rows into the DataTable and apply the changes to the database.

```csharp
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 );
recordsAffected = adapter.Update( dataTable );
```

7. Display the results of the updates.

```csharp
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

8. Close the connection.

```csharp
conn.Close();
```

To update rows using the SADDataAdapter object

1. Declare and initialize an SAConnection object.

```csharp
SAConnection conn = new SAConnection( c_connStr );
```

2. Open the connection.
conn.Open();

3. Create a new SADataAdapter object.

SADataAdapter adapter = new SADataAdapter();

4. Create an SACommand object and define its parameters.

The following code creates a SELECT and an UPDATE statement and defines the parameters for the UPDATE statement.

adapter.SelectCommand = new SACommand("SELECT * FROM Departments WHERE DepartmentID > 500", conn);
adapter.UpdateCommand = new SACommand("UPDATE Departments SET DepartmentName = ?, WHERE DepartmentID = ?", conn);
adapter.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;
SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add(parm);
parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
adapter.UpdateCommand.Parameters.Add(parm);

5. Fill the DataTable with the results of the SELECT statement.

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

6. Update the DataTable with the updated values for the rows and apply the changes to the database.

foreach (DataRow row in dataTable.Rows)
{
    row[1] = (string)row[1] + "_Updated";
}
recordsAffected = adapter.Update(dataTable);

7. Bind the results to the grid on the screen.

dataTable.Clear();
adapter.SelectCommand.CommandText = "SELECT * FROM Departments";
rowCount = adapter.Fill(dataTable);
dataGrid.DataSource = dataTable;

8. Close the connection.

conn.Close();

To delete rows from a table using the SADataAdapter object

1. Declare and initialize an SAConnection object.
SAConnection conn = new SAConnection( c_connStr );

2. Open the connection.

   conn.Open();

3. Create an SADataAdapter object.

   SADataAdapter adapter = new SADataAdapter();

4. Create the required SACommand objects and define any necessary parameters.

   The following code creates a SELECT and a DELETE statement and defines the parameters for the DELETE statement.

   adapter.SelectCommand = new SACommand(
       "SELECT * FROM Departments WHERE DepartmentID > 500",
       conn );
   adapter.DeleteCommand = new SACommand(
       "DELETE FROM Departments WHERE DepartmentID = ?",
       conn );
   adapter.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;
   SAParameter parm = new SAPparameter();
   parm.SAdbType = SAdbType.Integer;
   parm.SourceColumn = "DepartmentID";
   parm.SourceVersion = DataRowVersion.Original;
   adapter.DeleteCommand.Parameters.Add( parm );

5. Fill the DataTable with the results of the SELECT statement.

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

6. Modify the DataTable and apply the changes to the database.

   for each ( DataRow in dataTable.Rows )
   {
       row.Delete();
   }
   recordsAffected = adapter.Update( dataTable )

7. Bind the results to the grid on the screen.

   dataTable.Clear();
   rowCount = adapter.Fill( dataTable );
   dataGrid.DataSource = dataTable;

8. Close the connection.

   conn.Close();

Obtaining SADataAdapter schema information

When using the SADataAdapter, you can use the FillSchema method to obtain schema information about the result set in the DataSet. The FillSchema method returns the standard .NET DataTable object, which provides the names of all the columns in the result set.
To obtain DataSet schema information using the FillSchema method

1. Declare and initialize an SAConnection object.
   ```csharp
   SAConnection conn = new SAConnection(
       c_connStr);
   ```
2. Open the connection.
   ```csharp
   conn.Open();
   ```
3. Create an SADataAdapter with the SELECT statement you want to use. The schema is returned for the
   result set of this query.
   ```csharp
   SADataAdapter adapter = new SADataAdapter(
       "SELECT * FROM Employees", conn);
   ```
4. Create a new DataTable object, in this case called Table, to fill with the schema.
   ```csharp
   DataTable dataTable = new DataTable(
       "Table");
   ```
5. Fill the DataTable with the schema from the data source.
   ```csharp
   adapter.FillSchema( dataTable, SchemaType.Source );
   ```
6. Close the SAConnection object.
   ```csharp
   conn.Close();
   ```
7. Bind the DataSet to the grid on the screen.
   ```csharp
   dataGrid.DataSource = dataTable;
   ```

Obtaining 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.

When using the SADataAdapter, this technique can be used to fill the columns in the DataSet with the primary
key values generated by the data source. If you want to use this technique with the SACommand object, you
can either get the key columns from the parameters or reopen the DataReader.

Examples

The following examples use a table called adodotnet_primarykey that contains two columns, ID and Name.
The primary key for the table is ID. It is an INTEGER and contains an autoincremented value. The Name
column is CHAR(40).

These examples call the following stored procedure to retrieve the autoincremented primary key value from
the database.

```sql
CREATE 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
```
To insert a new row with an autoincremented primary key using the SACommand object

1. Connect to the database.

   ```csharp
   SAConnection conn = OpenConnection();
   ```

2. Create a new SACommand object to insert new rows into the DataTable. In the following code, the line
   ```csharp
   int id1 = (int) parmId.Value;
   ```
   verifies the primary key value of the row.

   ```csharp
   SACommand cmd = conn.CreateCommand();
   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();
   parmName.Value = "Marketing --- Command";
   cmd.ExecuteNonQuery();
   parmName.Value = "Sales --- Command";
   cmd.ExecuteNonQuery();
   parmName.Value = "Shipping --- Command";
   cmd.ExecuteNonQuery();
   ```

3. Bind the results to the grid on the screen and apply the changes to the database.

   ```csharp
   cmd.CommandText = "SELECT * FROM " +
   "adodotnet_primarykey";
   cmd.CommandType = CommandType.Text;
   SADataReader dr = cmd.ExecuteReader();
   dataGrid.DataSource = dr;
   ```

4. Close the connection.

   ```csharp
   conn.Close();
   ```

To insert a new row with an autoincremented primary key using the SADataAdapter object

1. Create a new SADataAdapter.

   ```csharp
   DataSet dataSet = new DataSet();
   SAConnection conn = OpenConnection();
   SADataAdapter adapter = new SADataAdapter();
   ```

2. Fill the data and schema of the DataSet. The SelectCommand is called by the SADataAdapter.Fill method to do this. You can also create the DataSet manually without using the Fill method and SelectCommand if you do not need the existing records.
3. Create a new SACommand to obtain the primary key values from the database.

```csharp
adapter.SelectCommand = new SACommand("select * from +
adodotnet_primarykey", conn );
```

```csharp
adapter.SelectCommand = new SACommand("sp_adodotnet_primarykey", conn);
adapter.SelectCommand.CommandType = CommandType.StoredProcedure;
adapter.SelectCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;
SAPassword parmId = new SAPassword();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
adapter.SelectCommand.Parameters.Add( parmId );
SAPassword parmName = new SAPassword();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "name";
parmName.SourceVersion = DataRowVersion.Current;
adapter.SelectCommand.Parameters.Add( parmName );
```

4. Fill the DataSet.

```csharp
adapter.Fill( dataSet );
```

5. Insert the new rows into the DataSet.

```csharp
DataRow row = dataSet.Tables[0].NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataSet.Tables[0].Rows.Add( row );
```

6. Apply the changes in the DataSet to the database. When the Update method is called, the primary key values are changed to the values obtained from the database.

```csharp
adapter.Update( dataSet );
dataGrid.DataSource = dataSet.Tables[0];
```

When you add new rows to the DataTable and call the Update method, the SDataAdapter calls the InsertCommand and maps the output parameters to the key columns for each new row. The Update method is called only once, but the InsertCommand is called by the Update method as many times as necessary for each new row being added.

7. Close the connection to the database.

```csharp
conn.Close();
```
Handling 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.

For more information, see “GetBytes method” on page 319 and “GetChars method” on page 321.

To issue a statement that returns a string using the GetChars method

1. Declare and initialize a Connection object.
2. Open the connection.
3. Add a Command object to define and execute a SQL statement.

   ```csharp
   SACommand cmd = new SACommand(
       "SELECT int_col, blob_col FROM test", conn);
   ```

4. Call the ExecuteReader method to return the DataReader object.

   ```csharp
   SADatareader reader = cmd.ExecuteReader();
   ```

   The following code reads the two columns from the result set. The first column is an integer
   (GetInt32(0)), while the second column is a LONG VARCHAR. GetChars is used to read 100
   characters at a time from the LONG VARCHAR column.

   ```csharp
   int length = 100;
   char[] buf = new char[length];
   int intValue;
   long dataIndex = 0;
   long charsRead = 0;
   long blobLength = 0;
   while( reader.Read() ) {
       intValue = reader.GetInt32( 0 );
       while ( ( charsRead = reader.GetChars(1, dataIndex, buf, 0, length ) ) == ( long )length ) {
           dataIndex += length;
       }
       blobLength = dataIndex + charsRead;
   }

5. Close the DataReader and Connection objects.

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

Obtaining 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. Using this method returns the data as a .NET Framework TimeSpan object.

For more information about the GetTimeSpan method, see “GetTimeSpan method” on page 334.
To convert a time value using the GetTimeSpan method

1. Declare and initialize a connection object.

   ```csharp
   SAConnection conn = new SAConnection(
       "Data Source=dsn-time-test;UID=DBA;PWD=sql" );
   ```

2. Open the connection.

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

3. Add a Command object to define and execute a SQL statement.

   ```csharp
   SACommand cmd = new SACommand(
       "SELECT ID, time_col FROM time_test", conn )
   ```

4. Call the ExecuteReader method to return the DataReader object.

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

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

   ```csharp
   while ( reader.Read() )
   {
       int ID = reader.GetInt32();
       TimeSpan time = reader.GetTimeSpan();
   }
   ```

5. Close the DataReader and Connection objects.

   ```csharp
   reader.Close();
   conn.Close();
   ```
Using stored procedures

You can use stored procedures with the .NET Data Provider. The ExecuteReader method is used to call stored procedures that return a result set, while the ExecuteNonQuery method is used to call stored procedures that do not return a result set. The ExecuteScalar method is used to call stored procedures that return only a single value.

When you call a stored procedure, you must create an SAPParameter object. Use a question mark as a placeholder for parameters, as follows:

```
sp_producttype( ?, ? )
```

For more information about the Parameter object, see “SAPParameter class” on page 382.

To execute a stored procedure

1. Declare and initialize an SAConnection object.

   ```csharp
   SAConnection conn = new SAConnection(
       "Data Source=SQL Anywhere 11 Demo" );
   ```

2. Open the connection.

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

3. Add an SACommand object to define and execute a SQL statement. The following code uses the CommandType property to identify the statement as a stored procedure.

   ```csharp
   SACommand cmd = new SACommand( "ShowProductInfo",
       conn );
   cmd.CommandType = CommandType.StoredProcedure;
   ```

   If you do not specify the CommandType property, then you must use a question mark as a placeholder for parameters, as follows:

   ```csharp
   SACommand cmd = new SACommand( 
       "call ShowProductInfo(?)", conn );
   cmd.CommandType = CommandType.Text;
   ```

4. Add an SAPParameter object to define the parameters for the stored procedure. You must create a new SAPParameter object for each parameter the stored procedure requires.

   ```csharp
   SAPParameter param = cmd.CreateParameter();
   param.SADbType = SADbType.Int32;
   param.Direction = ParameterDirection.Input;
   param.Value = 301;
   cmd.Parameters.Add( param );
   ```

   For more information about the Parameter object, see “SAPParameter class” on page 382.

5. Call the ExecuteReader method to return the DataReader object. The Get methods are used to return the results in the specified data type.

   ```csharp
   SADataReader reader = cmd.ExecuteReader();
   reader.Read();
   int ID = reader.GetInt32(0);
   string name = reader.GetString(1);
   ```
string descrip = reader.GetString(2);
decimal price = reader.GetDecimal(6);

6. Close the SADataReader and SAConnection objects.

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

Alternative way to call a stored procedure

Step 3 in the above instructions presents two ways you can call a stored procedure. Another way you can call a stored procedure, without using a Parameter object, is to call the stored procedure from your source code, as follows:

   SACommand cmd = new SACommand(
      "call ShowProductInfo(301)", conn);

For information about calling stored procedures that return a result set or a single value, see “Getting data using the SACommand object” on page 117.

For information about calling stored procedures that do not return a result set, see “Inserting, updating, and deleting rows using the SACommand object” on page 120.
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.

For more information about the SATransaction object, see “SATransaction class” on page 439.

Setting the isolation level for transactions

The database isolation level is used by default for transactions. However, 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.

For more information about isolation levels, see “Isolation levels and consistency” [SQL Anywhere Server - SQL Usage].

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

For more information about locking and isolation levels, see “Locking during queries” [SQL Anywhere Server - SQL Usage].

The following example uses an SATransaction object to issue and then roll back a SQL statement. The transaction uses isolation level 2 (RepeatableRead), which places a write lock on the row being modified so that no other database user can update the row.

To use an SATransaction object to issue a statement

1. Declare and initialize an SAConnection object.

   ```csharp
   SAConnection conn = new SAConnection(
       "Data Source=SQL Anywhere 11 Demo" );
   ```

2. Open the connection.

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

3. Issue a SQL statement to change the price of Tee shirts.

   ```csharp
   string stmt = "UPDATE Products SET UnitPrice = 2000.00 WHERE name = 'Tee shirt';"
   ```

4. Create an SATransaction object to issue the SQL statement using a Command object.

   Using a transaction allows you to specify the isolation level. Isolation level 2 (RepeatableRead) is used in this example so that another database user cannot update the row.
SATransaction trans = conn.BeginTransaction(IsolationLevel.RepeatableRead);
SACmd cmd = new SACmd(stmt, conn, trans);
int rows = cmd.ExecuteNonQuery();

5. Roll back the changes.
   trans.Rollback();

The SATransaction object allows you to commit or roll back your changes to the database. If you do not use a transaction, the .NET Data Provider operates in autocommit mode and you cannot roll back any changes that you make to the database. If you want to make the changes permanent, you would use the following:
   trans.Commit();

6. Close the SAConnection object.
   conn.Close();

**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 2.0 Data Provider. Distributed transaction has significant performance overhead. It is recommended that you use database transactions for non-distributed transactions.
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 throws SAException objects 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.

For more information about handling conflicts, see “Resolving conflicts when using the SADataAdapter” on page 125.

.NET Data Provider error handling example

The following example is from the Simple sample project. 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:

```csharp
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");
}
```

Connection 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:

```csharp
try {
    SqlConnection _conn = new SqlConnection( txtConnectionString.Text );
    _conn.Open();
} catch( SystemException ex ) {
    MessageBox.Show( ex.Message + " (" + ex.Source + ")", "Failed to connect");
}
```

For more error handling examples, see “Understanding the Simple sample project” on page 148 and “Understanding the Table Viewer sample project” on page 152.

For more information about error handling, see “SAFactory class” on page 359 and “SAError class” on page 349.
Deploying the SQL Anywhere .NET Data Provider

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 consists of two DLLs for each platform.

Windows required file

For Windows (except Windows Mobile), the following DLL is required:

- \install-dir\Assembly\v2\iAnywhere.Data.SQLAnywhere.dll

The file iAnywhere.Data.SQLAnywhere.dll is the DLL that is referenced by Visual Studio projects. The DLL is required for .NET Framework version 2.0 or later applications.

Windows Mobile required files

For Windows Mobile, the following DLL is required:

- \install-dir\ce\Assembly\v2\iAnywhere.Data.SQLAnywhere.dll

The file iAnywhere.Data.SQLAnywhere.dll is the DLL that is referenced by Visual Studio projects. 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 need to copy the Data Provider DLL to the device along with your program. It can go in the same directory as your application, or in the Windows directory.

Registering the SQL Anywhere .NET Data Provider DLL

The SQL Anywhere .NET Data Provider DLL (install-dir\Assembly\v2\iAnywhere.Data.SQLAnywhere.dll) needs to be registered in the Global Assembly Cache on Windows (except Windows Mobile). The Global Assembly Cache lists all the registered programs on your computer. When you install the .NET Data Provider,

Provider, the .NET Data Provider installation program registers it. On Windows Mobile, you do not need to register the DLL.

If you are deploying the .NET Data Provider, you must register the .NET Data Provider DLL ((install-dir \Assembly\v2\iAnywhere.Data.SQLAnywhere.dll) using the `gacutil` utility that is included with the .NET Framework.
Tracing support

The SQL Anywhere .NET provider supports tracing using the .NET 2.0 or later 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. Here's an example of the configuration file:

```xml
<?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"/>
        </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>
```

The trace configuration information is placed in the application's `bin\debug` folder under the name `app.exe.config`.

The `traceOutputOptions` that can be specified 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.
- **Threadld** 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.

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** The Trace All switch. When specified, 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.

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

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

  ```xml
  <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#
  ```

You can try application tracing using the TableViewer sample.

**To configure an application for tracing**

1. You must use .NET 2.0 or later.

   Start Visual Studio and open the TableViewer project file (TableViewer.sln) in samples-dir \SQLAnywhere\ADO.NET\TableViewer.
2. Place a copy of the configuration file shown above in the application's bin\debug folder under the name TableViewer.exe.config.

3. From the Debug menu, select Start Debugging.

When the application finishes execution, you will find a trace output file in samples-dir\SQLAnywhere \ADO.NET\TableViewer\bin\Debug\myTrace.log.

Tracing is not supported on Windows Mobile.

Tutorial: Using the SQL Anywhere .NET Data Provider

Contents

Introduction to the .NET Data Provider tutorial .......................................................... 146
Using the Simple code sample .................................................................................. 147
Using the Table Viewer code sample ........................................................................ 150
Introduction to the .NET Data Provider tutorial

This chapter 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.
Using the Simple code sample

This tutorial is based on the Simple project that is included with SQL Anywhere.

The complete application can be found in your SQL Anywhere samples directory at `samples-dir \SQLAnywhere\ADO.NET\SimpleWin32`.

For information about the default location of `samples-dir`, see “Samples directory” [SQL Anywhere Server - Database Administration].

The Simple project illustrates the following features:

- connecting to a database using the SAConnection object
- executing a query using the SACommand object
- obtaining the results using the SDataReader object
- basic error handling

For more information about how the sample works, see “Understanding the Simple sample project” on page 148.

To run the Simple code sample in Visual Studio

2. Choose File » Open » Project.
3. Browse to `samples-dir\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 DLL. This has already been done in the Simple code sample. You can view the reference to the Data Provider DLL in the following location:
   - In the Solution Explorer window, open the References folder.
   - You should see iAnywhere.Data.SQLAnywhere in the list.
     For instructions about adding a reference to the Data Provider DLL, see “Adding a reference to the Data Provider DLL in your project” on page 112.
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 choose View Code.
   - In the using directives in the top section, you should see the following line:
     ```csharp
     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. Choose 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 last name of each employee in the window, as follows:

8. Click the X in the upper right corner of the screen to shut down the application and disconnect from the sample database. This also shuts down the database server.

You have now run the application. The next section describes the application code.

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 11 - 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 samples-dir\SQLAnywhere\ADO.NET\SimpleWin32.

Declaring controls

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

```csharp
private System.Windows.Forms.ListBox listEmployees;
```

Connecting to the database

The btnConnect_Click method declares and initializes an SAConnection connection object.

```csharp
private void btnConnect_Click(object sender,
   System.EventArgs e)
   SAConnection conn = new SAConnection(
   "Data Source=SQL Anywhere 11 Demo;UID=DBA;PWD=sql");
```
The SAConnection 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 SAConnection object, see “SAConnection class” on page 246.

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

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

For more information about the SACommand object, see “SACommand class” on page 206.

**Displaying the results**  The results of the query are obtained using an SADDataReader object. The following code declares and creates an SADDataReader 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 SADDataReader.

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

The following code loops through the rows held in the SADDataReader 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 SADDataReader object, see “SADDataReader class” on page 310.

**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” on page 355.
Using the Table Viewer code sample

This tutorial is based on the Table Viewer project that is included with the SQL Anywhere .NET Data Provider.

The complete application can be found in your SQL Anywhere samples directory in samples-dir \SQLAnywhere\ADO.NET\TableViewer.

For information about the default location of samples-dir, see “Samples directory” [SQL Anywhere Server - Database Administration].

The Table Viewer project is more complex than the Simple project. It illustrates the following features:

● connecting to a database using the SAConnection object
● executing a query using the SACommand object
● obtaining the results using the SADataReader object
● using a grid to display the results using the DataGrid object
● more advanced error handling and result checking

For more information about how the sample works, see “Understanding the Table Viewer sample project” on page 152.

To run the Table Viewer code sample in Visual Studio

2. Choose File » Open » Project.
3. Browse to samples-dir\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. You can view the reference to the Data Provider DLL in the following location:
   ● In the Solution Explorer window, open the References folder.
   ● You should see iAnywhere.Data.SQLAnywhere in the list.
     For instructions about adding a reference to the Data Provider DLL, see “Adding a reference to the Data Provider DLL in your project” on page 112.
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 choose 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. Choose 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. In the upper right corner of the window, click X to shut down the application and disconnect from the SQL Anywhere sample database. This also shuts down the database server.

You have now run the application. The next section describes the application code.
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 11 - 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 samples-dir\SQLAnywhere\ADO.NET\TableViewer.

Declaring controls

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

```csharp
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox txtConnectString;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox txtSQLStatement;
private System.Windows.Forms.DataGrid dgResults;
```

Declaring a connection object

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

```csharp
private SAConnection _conn;
```

For more information about the SAConnection class, see “SAConnection class” on page 246.

Connecting to the database

The Text property of the txtConnectString object has a default value of "Data Source=SQL Anywhere 11 Demo". This value can be overridden by the application user by typing a new value into the txtConnectString 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.

```csharp
this.txtConnectString.Text = "Data Source=SQL Anywhere 11 Demo";
```

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

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

For more information about the SAConnection constructor, see “SAConnection members” on page 247.

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 SACommand object. The following code declares and creates a
command object using the SACommand constructor. This constructor accepts a string representing the query
to be executed, along with the SAConnection object that represents the connection that the query is executed
on.

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

For more information about the SACommand object, see “SACommand class” on page 206.

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

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

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

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

For more information about the SADrReader object, see “SADrReader class” on page 310.

**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 SAConnection( txtConnectString.Text );
    _conn.Open();

    SACommand cmd = new SACommand(
        "SELECT table_name FROM SYS.SYSTAB where creator = 101", _conn );
    SADrReader   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” on page 355.
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.
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 reinstall 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.
Registering the connection string

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="ENG=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

```xml
<connectionStrings>
  <add name="MyConnectionString"
      connectionString="DSN=MyDataSource"
      providerName="iAnywhere.Data.SQLAnywhere"/>
</connectionStrings>
```
Registering the 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

```xml
<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 161.

XML code sample for Roles Provider registration

```xml
<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 162.
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 163.

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 164.

XML code sample for Health Monitoring Provider registration

```
<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 165.
# Membership provider XML attributes

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>type</td>
<td>iAnywhere.Web.Security.SAMembershipProvider</td>
</tr>
<tr>
<td>connectionStringName</td>
<td>The name of a connection string specified in the <code>&lt;connection-Strings&gt;</code> element.</td>
</tr>
<tr>
<td>connectionString</td>
<td>The connection string. Optional. Required if <code>connectionStringName</code> is not specified.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The application name with which to associate provider data.</td>
</tr>
<tr>
<td>commandTimeout</td>
<td>The timeout value, in seconds, for server calls.</td>
</tr>
<tr>
<td>enablePasswordReset</td>
<td>Valid entries are true or false.</td>
</tr>
<tr>
<td>enablePasswordRetrieval</td>
<td>Valid entries are true or false.</td>
</tr>
<tr>
<td>maxInvalidPasswordAttempts</td>
<td>Valid entries are true or false.</td>
</tr>
<tr>
<td>minRequiredNonalphanumericCharacters</td>
<td>The minimum number of special characters that must be present in a valid password.</td>
</tr>
<tr>
<td>minRequiredPasswordLength</td>
<td>The minimum length required for a password.</td>
</tr>
<tr>
<td>passwordAttemptWindow</td>
<td>The time window between which consecutive failed attempts to provide a valid password or password answer are tracked.</td>
</tr>
<tr>
<td>passwordFormat</td>
<td>Valid entries are Clear, Hashed, or Encrypted.</td>
</tr>
<tr>
<td>requiresQuestionAndAnswer</td>
<td>Valid entries are true or false.</td>
</tr>
<tr>
<td>requiresUniqueEmail</td>
<td>Valid entries are true or false.</td>
</tr>
<tr>
<td>passwordStrengthRegularExpression</td>
<td>The regular expression used to evaluate a password.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>type</td>
<td>iAnywhere.Web.Security.SARoleProvider</td>
</tr>
<tr>
<td>connectionString-Name</td>
<td>The name of a connection string specified in the &lt;connectionStrings&gt; element.</td>
</tr>
<tr>
<td>connectionString</td>
<td>The connection string. Optional. Required if connectionStringName is not specified.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The application name with which to associate provider data.</td>
</tr>
<tr>
<td>commandTimeout</td>
<td>The timeout value, in seconds, for server calls.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>type</td>
<td>iAnywhere.Web.Security.SAProfileProvider</td>
</tr>
<tr>
<td>connectionString-Name</td>
<td>The name of a connection string specified in the <code>&lt;connectionStrings&gt;</code> element.</td>
</tr>
<tr>
<td>connectionString</td>
<td>The connection string. Optional. Required if connectionStringName is not specified.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The application name with which to associate provider data.</td>
</tr>
<tr>
<td>commandTime-out</td>
<td>The timeout value, in seconds, for server calls.</td>
</tr>
</tbody>
</table>
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.

SARoleProvider 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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>type</td>
<td>iAnywhere.Web.Security.SAPersonalizationProvider</td>
</tr>
<tr>
<td>connectionString-Name</td>
<td>The name of a connection string specified in the &lt;connectionStrings&gt; element.</td>
</tr>
<tr>
<td>connectionString</td>
<td>The connection string. Optional. Required if connectionStringName is not specified.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The application name with which to associate provider data.</td>
</tr>
<tr>
<td>commandTimeout</td>
<td>The timeout value, in seconds, for server calls.</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the provider.</td>
</tr>
<tr>
<td>type</td>
<td>iAnywhere.Web.Security.SAWebEventProvider</td>
</tr>
<tr>
<td>connectionString-Name</td>
<td>The name of a connection string specified in the <code>&lt;connectionStrings&gt;</code> element.</td>
</tr>
<tr>
<td>connectionString</td>
<td>The connection string. Optional. Required if connectionStringName is not specified.</td>
</tr>
<tr>
<td>commandTimeout</td>
<td>The timeout value, in seconds, for server calls.</td>
</tr>
<tr>
<td>maxEventDetailsLength</td>
<td>The maximum length of the details string for each event or Infinite</td>
</tr>
</tbody>
</table>
Tutorial: Developing a simple .NET database application with Visual Studio

Contents

Lesson 1: Create a table viewer ................................................................. 168
Lesson 2: Add a synchronizing data control ............................................. 172
Lesson 1: Create 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 samples-dir\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.

To develop a database application with Visual Studio

2. From the Visual Studio File menu, choose New » Project.
   The New Project window appears.
   a. In the left pane of the New Project window, choose either Visual Basic or Visual C# for the programming language.
   b. From the Windows subcategory, choose 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. In the Visual Studio View menu, choose Server Explorer.
4. In the Server Explorer window, right-click Data Connections and choose Add Connection.
   A new connection named SQL Anywhere.demo11 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 will see a list of data sources. Choose 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, choose ODBC Data Source Name and type SQL Anywhere 11 Demo.
c. Click **Test Connection** to verify that you can connect to the sample database.

d. Click **OK**.

6. Expand the **SQL Anywhere.demo11** connection in the **Server Explorer** window until you see the table names.
   
   a. Right-click the Products table and choose **Show Table Data**.
      
      This shows the rows and columns of the Products table in a window.
   
   b. Close the table data window.

7. From the Visual Studio **Data** menu, choose **Add New Data Source**.

8. In the **Data Source Configuration Wizard**, do the following:
   
   a. On the **Data Source Type** page, choose **Database** and click **Next**.
   
   b. On the **Data Connection** page, choose **SQL Anywhere.demo11** and 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, choose **Tables** and click **Finish**.

9. From the Visual Studio **Data** menu, choose **Show Data Sources**. The **Data Sources** window appears.

Expand the Products table in the **Data Sources** window.

a. Click Products and choose **Details** from the dropdown list.

b. Click Photo and choose **Picture Box** from the dropdown list.

c. Click Products and drag it to your form (Form1).

10. On the form, choose 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, choose **Zoom**.

   d. To close the **Picture Box Tasks** window, click anywhere outside the window.

A dataset control and several labeled text fields appear on the form.
11. Build and run the project.
   a. From the Visual Studio Build menu, choose Build Solution.
   b. From the Visual Studio Debug menu, choose 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 diskette
   icon.

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: Add a synchronizing data control

This tutorial is a continuation of the tutorial described in “Lesson 1: Create a table viewer” on page 168. The complete application can be found in the ADO.NET project samples-dir\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.

To add a datagrid control

1. Start Visual Studio and load your MySimpleViewer project that you created in “Lesson 1: Create a table viewer” on page 168.
2. Right-click DataSet1 in the Data Sources window and choose Edit DataSet With Designer.
3. Right-click an empty area in the DataSet Designer window and choose 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, make sure Use SQL Statements is chosen and then click Next.
   c. On the Enter A SQL Statement page, click Query Builder.
   d. On the Add Table window, click the Views tab, choose ViewSalesOrders, and 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. Choose Region.
   c. Choose Quantity.
   d. Choose 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:

```sql
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 choose **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.
    - From the Visual Studio **Build** menu, choose **Build Solution**.
● From the Visual Studio Debug menu, choose 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 and choose 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 and choose Properties.
   b. Click the Events icon (it appears as a lightning bolt).
   c. Scroll down until you find the TextChanged event.
d. Click **TextChanged** and choose **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.

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 .NET 2.0 API reference

Contents

iAnywhere.Data.SQLAnywhere namespace (.NET 2.0) .................................................. 178
iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)

SABulkCopy class

Efficiently bulk load a SQL Anywhere table with data from another source. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SABulkCopy
    Implements IDisposable

C#

public sealed class SABulkCopy : IDisposable

Remarks

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.

Implements: IDisposable

See also

● “SABulkCopy members” on page 178

SABulkCopy members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SABulkCopy constructors</td>
<td>Initializes an SABulkCopy object.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchSize property</td>
<td>Gets or sets the number of rows in each batch. At the end of each batch, the rows in the batch are sent to the server.</td>
</tr>
<tr>
<td>BulkCopyTimeout property</td>
<td>Gets or sets the number of seconds for the operation to complete before it times out.</td>
</tr>
<tr>
<td>ColumnMappings property</td>
<td>Returns a collection of SABulkCopyColumnMapping items. Column mappings define the relationships between columns in the data source and columns in the destination.</td>
</tr>
</tbody>
</table>
### Member name

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationTableName property</td>
<td>Gets or sets the name of the destination table on the server.</td>
</tr>
<tr>
<td>NotifyAfter property</td>
<td>Gets or sets the number of rows to be processed before generating a notification event.</td>
</tr>
</tbody>
</table>

### Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Closes the SABulkCopy instance.</td>
</tr>
<tr>
<td>Dispose method</td>
<td>Disposes of the SABulkCopy instance.</td>
</tr>
<tr>
<td>WriteToServer methods</td>
<td>Copies all rows in the supplied array of DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.</td>
</tr>
</tbody>
</table>

### Public events

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SARowsCopied event</td>
<td>This event occurs every time the number of rows specified by the NotifyAfter property have been processed.</td>
</tr>
</tbody>
</table>

### See also
- “SABulkCopy class” on page 178

### SABulkCopy constructors

Initializes an SABulkCopy object.

### SABulkCopy(SAConnection) constructor

#### Syntax

**Visual Basic**

```vbnet
Public Sub New(ByVal connection As SAConnection)
```

**C#**

```csharp
public SABulkCopy(
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
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

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “SABulkCopy constructors” on page 179

SABulkCopy(String) constructor

Initializes an SABulkCopy object.

Syntax

Visual Basic

Public Sub New(_
    ByVal connectionString As String _
)"

C#

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.

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “SABulkCopy constructors” on page 179
SABulkCopy(String, SABulkCopyOptions) constructor

Initializes an SABulkCopy object.

Syntax

Visual Basic

Public Sub New(_
    ByVal connectionString As String, _
    ByVal copyOptions As SABulkCopyOptions _
)

C#

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.

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “SABulkCopy constructors” on page 179

SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction) constructor

Initializes an SABulkCopy object.

Syntax

Visual Basic

Public Sub New(_
    ByVal connectionString As SAConnection,
    ByVal copyOptions As SABulkCopyOptions, _
    ByVal externalTransaction As SATransaction _
)

C#
C# 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
**Restrictions:** The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also
- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “SABulkCopy constructors” on page 179

**BatchSize property**

Gets or sets the number of rows in each batch. At the end of each batch, the rows in the batch are sent to the server.

Syntax
- **Visual Basic**
  ```vbnet
  Public Property BatchSize As Integer
  ```
- **C#**
  ```csharp
  public int BatchSize { get; set; }
  ```

Property value
The number of rows in each batch. The default is 0.

Remarks
- 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.

See also

● “SABulkCopy class” on page 178
● “SABulkCopy members” on page 178

**BulkCopyTimeout property**

Gets or sets the number of seconds for the operation to complete before it times out.

**Syntax**

**Visual Basic**

Public Property **BulkCopyTimeout** As Integer

**C#**

public int **BulkCopyTimeout** { get; set; }

**Property value**

The default value is 30 seconds.

**Remarks**

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.

See also

● “SABulkCopy class” on page 178
● “SABulkCopy members” on page 178

**ColumnMappings property**

Returns a collection of SABulkCopyColumnMapping items. Column mappings define the relationships between columns in the data source and columns in the destination.

**Syntax**

**Visual Basic**

Public Readonly Property **ColumnMappings** As SABulkCopyColumnMappingCollection
C#

    public SABulkCopyColumnMappingCollection ColumnMappings { get; }

Property value
By default, it is an empty collection.

Remarks
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.

See also
- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178

DestinationTableName property

Gets or sets the name of the destination table on the server.

Syntax

Visual Basic

    Public Property DestinationTableName As String

C#

    public string DestinationTableName { get; set; }

Property value
The default value is a null reference. In Visual Basic it is Nothing.

Remarks
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.

See also
- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
**NotifyAfter property**

Gets or sets the number of rows to be processed before generating a notification event.

**Syntax**

**Visual Basic**

```
Public Property NotifyAfter As Integer
```

**C#**

```
public int NotifyAfter { get; set; }
```

**Property value**

Zero is returned if the property has not been set.

**Remarks**

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 class” on page 178
- “SABulkCopy members” on page 178
- “BulkCopyTimeout property” on page 183

**Close method**

Closes the SABulkCopy instance.

**Syntax**

**Visual Basic**

```
Public Sub Close()
```

**C#**

```
public void Close();
```

**See also**

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
Dispose method

Disposes of the SABulkCopy instance.

Syntax

Visual Basic

NotOverridable Public Sub Dispose()

C#

public void Dispose();

See also

● “SABulkCopy class” on page 178
● “SABulkCopy members” on page 178

WriteToServer methods

Copies all rows in the supplied array of DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.

WriteToServer(DataRow[]) method

Copies all rows in the supplied array of DataRow objects to a destination table specified by the DestinationTableName property of the SABulkCopy object.

Syntax

Visual Basic

Public Sub WriteToServer(_
    ByVal rows As DataRow() _
) _

C#

public void WriteToServer(
    DataRow[] rows
);  

Parameters

● rows An array of System.Data.DataRow objects that will be copied to the destination table.

Remarks

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.
WriteToServer(DataTable) method

Copies all rows in the supplied DataTable to a destination table specified by the DestinationTableName property of the SABulkCopy object.

Syntax

Visual Basic

Public Sub WriteToServer( _
    ByVal table As DataTable _
)  

C#

public void WriteToServer( 
    DataTable table 
);  

Parameters

- **table** A System.Data.DataTable whose rows will be copied to the destination table.

Remarks

**Restrictions:** The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “WriteToServer methods” on page 186
- “DestinationTableName property” on page 184

WriteToServer(IDataReader) method

Copies all rows in the supplied IDataReader to a destination table specified by the DestinationTableName property of the SABulkCopy object.

Syntax

Visual Basic

Public Sub WriteToServer( _
    ByVal reader As IDataReader _
)
C#

    public void WriteToServer(
    IDataReader reader
    );

Parameters

- **reader**  A System.Data.IDataReader whose rows will be copied to the destination table.

Remarks

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “WriteToServer methods” on page 186
- “DestinationTableName property” on page 184

### WriteToServer(DataTable, DataRowState) method

Copies all rows in the supplied DataTable with the specified row state to a destination table specified by the DestinationTableName property of the SABulkCopy object.

Syntax

**Visual Basic**

    Public Sub WriteToServer( _
    ByVal table As DataTable, _
    ByVal rowState As DataRowState _
    )

**C#**

    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.

Restrictions: The SABulkCopy class is not available in the .NET Compact Framework 2.0.
See also

- “SABulkCopy class” on page 178
- “SABulkCopy members” on page 178
- “WriteToServer methods” on page 186
- “DestinationTableName property” on page 184

### SARowsCopied event

This event occurs every time the number of rows specified by the NotifyAfter property have been processed.

**Syntax**

**Visual Basic**

```vbnet
Public Event SARowsCopied As SARowsCopiedEventHandler
```

**C#**

```csharp
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 class” on page 178
- “SABulkCopy members” on page 178
- “NotifyAfter property” on page 185

### SABulkCopyColumnMapping class

Defines the mapping between a column in an SABulkCopy instance's data source and a column in the instance's destination table. This class cannot be inherited.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SABulkCopyColumnMapping
```

**C#**

```csharp
public sealed class SABulkCopyColumnMapping
```

**Remarks**

**Restrictions:** The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.
See also
- “SABulkCopyColumnMapping members” on page 190

**SABulkCopyColumnMapping members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SABulkCopyColumnMapping constructors</td>
<td>Initializes a new instance of the “SABulkCopyColumnMapping class” on page 189.</td>
</tr>
</tbody>
</table>

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationColumn property</td>
<td>Gets or sets the name of the column in the destination database table being mapped to.</td>
</tr>
<tr>
<td>DestinationOrdinal property</td>
<td>Gets or sets the ordinal value of the column in the destination table being mapped to.</td>
</tr>
<tr>
<td>SourceColumn property</td>
<td>Gets or sets the name of the column being mapped in the data source.</td>
</tr>
<tr>
<td>SourceOrdinal property</td>
<td>Gets or sets ordinal position of the source column within the data source.</td>
</tr>
</tbody>
</table>

See also
- “SABulkCopyColumnMapping class” on page 189

**SABulkCopyColumnMapping constructors**

Initializes a new instance of the “SABulkCopyColumnMapping class” on page 189.

**SABulkCopyColumnMapping() constructor**

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

**Syntax**

**Visual Basic**

```
Public Sub New()
```
C#

public SABulkCopyColumnMapping();

Remarks

Restrictions: The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

See also

● “SABulkCopyColumnMapping class” on page 189
● “SABulkCopyColumnMapping members” on page 190
● “SABulkCopyColumnMapping constructors” on page 190

SABulkCopyColumnMapping(Int32, Int32) constructor

Creates a new column mapping, using column ordinals to refer to source and destination columns.

Syntax

Visual Basic

Public Sub New(
    ByVal sourceColumnOrdinal As Integer, 
    ByVal destinationColumnOrdinal As Integer 
)

C#

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

Restrictions: The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

See also

● “SABulkCopyColumnMapping class” on page 189
● “SABulkCopyColumnMapping members” on page 190
● “SABulkCopyColumnMapping constructors” on page 190
SABulkCopyColumnMapping(Int32, 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.

Syntax

Visual Basic

Public Sub New(
    ByVal sourceColumnOrdinal As Integer, _
    ByVal destinationColumn As String _
)

C#

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

Restrictions: The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopyColumnMapping class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “SABulkCopyColumnMapping constructors” on page 190

SABulkCopyColumnMapping(String, Int32) 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.

Syntax

Visual Basic

Public Sub New(
    ByVal sourceColumn As String, _
    ByVal destinationColumnOrdinal As Integer _
)


C#

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

**Restrictions:** The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopyColumnMapping class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “SABulkCopyColumnMapping constructors” on page 190

SABulkCopyColumnMapping(String, String) constructor

Creates a new column mapping, using column names to refer to source and destination columns.

Syntax

Visual Basic

Public Sub New(
    ByVal sourceColumn As String, _
    ByVal destinationColumn As String _
)

C#

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

**Restrictions:** The SABulkCopyColumnMapping class is not available in the .NET Compact Framework 2.0.
See also

- “SABulkCopyColumnMapping class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “SABulkCopyColumnMapping constructors” on page 190

DestinationColumn property

Gets or sets the name of the column in the destination database table being mapped to.

Syntax

**Visual Basic**

```vbnet
Public Property DestinationColumn As String
```

**C#**

```csharp
public string DestinationColumn { get; set; }
```

Property value

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.

Remarks

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 class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “DestinationOrdinal property” on page 194

DestinationOrdinal property

Gets or sets the ordinal value of the column in the destination table being mapped to.

Syntax

**Visual Basic**

```vbnet
Public Property DestinationOrdinal As Integer
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
C#

    public int DestinationOrdinal { get; set; }

**Property value**

An integer specifying the ordinal of the column being mapped to in the destination table or -1 if the property is not set.

**Remarks**

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 class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “DestinationColumn property” on page 194

**SourceColumn property**

Gets or sets the name of the column being mapped in the data source.

**Syntax**

**Visual Basic**

    Public Property SourceColumn As String

**C#**

    public string SourceColumn { get; set; }

**Property value**

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.

**Remarks**

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.
SourceOrdinal property

Gets or sets ordinal position of the source column within the data source.

Syntax

Visual Basic

Public Property SourceOrdinal As Integer

C#

public int SourceOrdinal { get; set; }

Property value

An integer specifying the ordinal of the column in the data source or -1 if the property is not set.

Remarks

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 class” on page 189
- “SABulkCopyColumnMapping members” on page 190
- “SourceColumn property” on page 195

SABulkCopyColumnMappingCollection class

A collection of SABulkCopyColumnMapping objects that inherits from System.Collections.CollectionBase. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SABulkCopyColumnMappingCollection
Inherits CollectionBase
C#

public sealed class SABulkCopyColumnMappingCollection : CollectionBase

Remarks

Restrictions: The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopyColumnMappingCollection members” on page 197

SABulkCopyColumnMappingCollection members

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity (inherited from CollectionBase)</td>
<td>Gets or sets the number of elements that the CollectionBase can contain.</td>
</tr>
<tr>
<td>Count (inherited from CollectionBase)</td>
<td>Gets the number of elements contained in the CollectionBase instance. This property cannot be overridden.</td>
</tr>
<tr>
<td>Item property</td>
<td>Gets the SABulkCopyColumnMapping object at the specified index.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add methods</td>
<td>Adds the specified SABulkCopyColumnMapping object to the collection.</td>
</tr>
<tr>
<td>Clear (inherited from CollectionBase)</td>
<td>Removes all objects from the CollectionBase instance. This method cannot be overridden.</td>
</tr>
<tr>
<td>Contains method</td>
<td>Gets a value indicating whether a specified SABulkCopyColumnMapping object exists in the collection.</td>
</tr>
<tr>
<td>CopyTo method</td>
<td>Copies the elements of the SABulkCopyColumnMappingCollection to an array of SABulkCopyColumnMapping items, starting at a particular index.</td>
</tr>
<tr>
<td>GetEnumerator (inherited from CollectionBase)</td>
<td>Returns an enumerator that iterates through the CollectionBase instance.</td>
</tr>
<tr>
<td>IndexOf method</td>
<td>Gets or sets the index of the specified SABulkCopyColumnMapping object within the collection.</td>
</tr>
</tbody>
</table>
### Member name | Description
---|---
Remove method | Removes the specified SABulkCopyColumnMapping element from the SABulkCopyColumnMappingCollection.
RemoveAt method | Removes the mapping at the specified index from the collection.

**See also**
- “SABulkCopyColumnMappingCollection class” on page 196

## Item property

**Description**

Gets the SABulkCopyColumnMapping object at the specified index.

**Syntax**

**Visual Basic**

```vbnet
Public Readonly Property Item ( _
  ByVal index As Integer _) As SABulkCopyColumnMapping
```

**C#**

```csharp
public SABulkCopyColumnMapping this [ int index ] { get; }
```

**Parameters**

- **index** The zero-based index of the SABulkCopyColumnMapping object to find.

**Property value**

An SABulkCopyColumnMapping object is returned.

**See also**

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197

## Add methods

**Description**

Adds the specified SABulkCopyColumnMapping object to the collection.

### Add(SABulkCopyColumnMapping) method

Adds the specified SABulkCopyColumnMapping object to the collection.
Syntax

Visual Basic

Public Function Add( _
    ByVal bulkCopyColumnMapping As SABulkCopyColumnMapping _
) As SABulkCopyColumnMapping

C#

public SABulkCopyColumnMapping Add(
    SABulkCopyColumnMapping bulkCopyColumnMapping
);

Parameters

- **bulkCopyColumnMapping** The SABulkCopyColumnMapping object that describes the mapping to be added to the collection.

Remarks

Restrictions: The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197
- “Add methods” on page 198
- “SABulkCopyColumnMapping class” on page 189

Add(Int32, Int32) method

Creates a new SABulkCopyColumnMapping object using ordinals to specify both source and destination columns, and adds the mapping to the collection.

Syntax

Visual Basic

Public Function Add( _
    ByVal sourceColumnOrdinal As Integer, _
    ByVal destinationColumnOrdinal As Integer _
) As SABulkCopyColumnMapping

C#

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**

**Restrictions:** The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

**See also**

• “SABulkCopyColumnMappingCollection class” on page 196
• “SABulkCopyColumnMappingCollection members” on page 197
• “Add methods” on page 198

**Add(Int32, 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.

**Syntax**

**Visual Basic**

```vbnet
Public Function Add( _
    ByVal sourceColumnOrdinal As Integer, _
    ByVal destinationColumn As String _) As SABulkCopyColumnMapping
```

**C#**

```csharp
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**

**Restrictions:** The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

**See also**

• “SABulkCopyColumnMappingCollection class” on page 196
• “SABulkCopyColumnMappingCollection members” on page 197
• “Add methods” on page 198
Add(String, Int32) 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.

Creates a new column mapping, using column ordinals or names to refer to source and destination columns.

Syntax

Visual Basic

Public Function Add(  
    ByVal sourceColumn As String,  
    ByVal destinationColumnOrdinal As Integer  
) As SABulkCopyColumnMapping

C#

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

Restrictions: The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

See also

● “SABulkCopyColumnMappingCollection class” on page 196

● “SABulkCopyColumnMappingCollection members” on page 197

● “Add methods” on page 198

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.

Syntax

Visual Basic

Public Function Add(  
    ByVal sourceColumn As String,  
    ByVal destinationColumn As String  
) As SABulkCopyColumnMapping
C#

```csharp
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**

**Restrictions:** The SABulkCopyColumnMappingCollection class is not available in the .NET Compact Framework 2.0.

**See also**

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197
- “Add methods” on page 198

**Contains method**

Gets a value indicating whether a specified SABulkCopyColumnMapping object exists in the collection.

**Syntax**

**Visual Basic**

```vbnet
Public Function Contains(  
    ByVal value As SABulkCopyColumnMapping  
) As Boolean
```

**C#**

```csharp
public bool Contains(  
    SABulkCopyColumnMapping value  
);
```

**Parameters**

- **value**  A valid SABulkCopyColumnMapping object.

**Return value**

True if the specified mapping exists in the collection; otherwise, false.

**See also**

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197
CopyTo method

Copies the elements of the SABulkCopyColumnMappingCollection to an array of SABulkCopyColumnMapping items, starting at a particular index.

Syntax

Visual Basic

Public Sub CopyTo(  
    ByVal array As SABulkCopyColumnMapping(),  
    ByVal index As Integer  
)

C#

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.

See also

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197

IndexOf method

Gets or sets the index of the specified SABulkCopyColumnMapping object within the collection.

Syntax

Visual Basic

Public Function IndexOf(  
    ByVal value As SABulkCopyColumnMapping  
) As Integer

C#

public int IndexOf(  
    SABulkCopyColumnMapping value  
);

Parameters

- **value** The SABulkCopyColumnMapping object to search for.
Return value

The zero-based index of the column mapping is returned, or -1 is returned if the column mapping is not found in the collection.

See also

● “SABulkCopyColumnMappingCollection class” on page 196
● “SABulkCopyColumnMappingCollection members” on page 197

Remove method

Removes the specified SABulkCopyColumnMapping element from the SABulkCopyColumnMappingCollection.

Syntax

Visual Basic

Public Sub Remove( _
    ByVal value As SABulkCopyColumnMapping _
) _

C#

public void Remove( _
    SABulkCopyColumnMapping value _
); _

Parameters

● value The SABulkCopyColumnMapping object to be removed from the collection.

See also

● “SABulkCopyColumnMappingCollection class” on page 196
● “SABulkCopyColumnMappingCollection members” on page 197

RemoveAt method

Removes the mapping at the specified index from the collection.

Syntax

Visual Basic

Public Sub RemoveAt( _
    ByVal index As Integer _
) _

C#

public void RemoveAt( _
    index _
) _

Parameters

- **index**  
The zero-based index of the SABulkCopyColumnMapping object to be removed from the collection.

See also

- “SABulkCopyColumnMappingCollection class” on page 196
- “SABulkCopyColumnMappingCollection members” on page 197

**SABulkCopyOptions enumeration**

A bitwise flag that specifies one or more options to use with an instance of SABulkCopy.

Syntax

**Visual Basic**

```vbnet
Public Enum SABulkCopyOptions

C#

```public enum SABulkCopyOptions

Remarks

The SABulkCopyOptions enumeration is used when you construct an SABulkCopy object to specify how the WriteToServer methods will behave.

**Restrictions:** The SABulkCopyOptions class is not available in the .NET Compact Framework 2.0.

The CheckConstraints and KeepNulls options are not supported.

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Specifying only this value causes the default behavior to be used. By default, triggers are enabled.</td>
<td>0</td>
</tr>
<tr>
<td>DoNotFireTriggers</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>KeepIdentity</td>
<td>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.</td>
<td>2</td>
</tr>
<tr>
<td>TableLock</td>
<td>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.</td>
<td>4</td>
</tr>
<tr>
<td>UseInternalTransaction</td>
<td>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.</td>
<td>8</td>
</tr>
</tbody>
</table>

See also

- “SABulkCopy class” on page 178

**SACommand class**

A SQL statement or stored procedure that is executed against a SQL Anywhere database. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SACommand
Inherits DbCommand
Implements ICloneable

**C#**

public sealed class SACommand : DbCommand, ICloneable

**Remarks**

Implements: ICloneable

For more information, see “Accessing and manipulating data” on page 117.

See also

- “SACommand members” on page 207
SACommand members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACommand constructors</td>
<td>Initializes a new instance of the “SACommand class” on page 206.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandText property</td>
<td>Gets or sets the text of a SQL statement or stored procedure.</td>
</tr>
<tr>
<td>CommandTimeout property</td>
<td>Gets or sets the wait time in seconds before terminating an attempt to execute a command and generating an error.</td>
</tr>
<tr>
<td>CommandType property</td>
<td>Gets or sets the type of command represented by an SACommand.</td>
</tr>
<tr>
<td>Connection property</td>
<td>Gets or sets the connection object to which the SACommand object applies.</td>
</tr>
<tr>
<td>DesignTimeVisible property</td>
<td>Gets or sets a value that indicates if the SACommand should be visible in a Windows Form Designer control. The default is true.</td>
</tr>
<tr>
<td>Parameters property</td>
<td>A collection of parameters for the current statement. Use question marks in the CommandText to indicate parameters.</td>
</tr>
<tr>
<td>Transaction property</td>
<td>Specifies the SATransaction object in which the SACommand executes.</td>
</tr>
<tr>
<td>UpdatedRowSource property</td>
<td>Gets or sets how command results are applied to the DataRow when used by the Update method of the SDataAdapter.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginExecuteNonQuery methods</td>
<td>Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.</td>
</tr>
<tr>
<td>BeginExecuteReader methods</td>
<td>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.</td>
</tr>
<tr>
<td>Cancel method</td>
<td>Cancels the execution of an SACommand object.</td>
</tr>
</tbody>
</table>
### CreateParameter method
Provides an SAParameter object for supplying parameters to SACommand objects.

### EndExecuteNonQuery method
Finishes asynchronous execution of a SQL statement or stored procedure.

### EndExecuteReader method
Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested SADataReader.

### ExecuteNonQuery method
Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data definition statement.

### ExecuteReader methods
Executes a SQL statement that returns a result set.

### ExecuteScalar method
Executes a statement that returns a single value. 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 SACommand on the data source.

### ResetCommandTimeout method
Resets the CommandTimeout property to its default value of 30 seconds.

---

**See also**
- “SACommand class” on page 206

---

## SACommand constructors

Initializes a new instance of the “SACommand class” on page 206.

### SACommand() constructor

Initializes an SACommand object.

**Syntax**

**Visual Basic**

Public Sub New()

**C#**

public SACommand();
See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SACommand constructors” on page 208

**SACommand(String) constructor**

Initializes an SACommand object.

Syntax

Visual Basic

```vbnet
Public Sub New(_
    ByVal cmdText As String _
)"
```

C#

```csharp
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.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SACommand constructors” on page 208

**SACommand(String, SAConnection) constructor**

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

Syntax

Visual Basic

```vbnet
Public Sub New(_
    ByVal cmdText As String, _
    ByVal connection As SAConnection _
)"
```

C#

```csharp
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.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SACommand constructors” on page 208

**SACommand(String, SAConnection, SATransaction) constructor**

A SQL statement or stored procedure that is executed against a SQL Anywhere database.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New(
  ByVal cmdText As String, 
  ByVal connection As SAConnection, 
  ByVal transaction As SATransaction
)
```

**C#**

```csharp
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

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SACommand constructors” on page 208
- “SATransaction class” on page 439
CommandText property

Gets or sets the text of a SQL statement or stored procedure.

Syntax

Visual Basic

Public Overrides Property CommandText As String

C#

public override string CommandText { get; set; }

Property value

The SQL statement or the name of the stored procedure to execute. The default is an empty string.

See also

● “SACommand class” on page 206
● “SACommand members” on page 207
● “SACommand() constructor” on page 208

CommandTimeout property

Gets or sets the wait time in seconds before terminating an attempt to execute a command and generating an error.

Syntax

Visual Basic

Public Overrides Property CommandTimeout As Integer

C#

public override int CommandTimeout { get; set; }

Property value

The default value is 30 seconds.

Remarks

A value of 0 indicates no limit. This should be avoided because it may cause the attempt to execute a command to wait indefinitely.

See also

● “SACommand class” on page 206
● “SACommand members” on page 207
**CommandType property**

Gets or sets the type of command represented by an SACommand.

**Syntax**

**Visual Basic**

Public Overrides Property **CommandType** As CommandType

**C#**

public override CommandType **CommandType** { get; set; }

**Property value**

One of the CommandType values. The default is CommandType.Text.

**Remarks**

Supported command types are as follows:

- **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.
- **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.

**See also**

- “SACommand class” on page 206
- “SACommand members” on page 207

**Connection property**

Gets or sets the connection object to which the SACommand object applies.

**Syntax**

**Visual Basic**

Public Property **Connection** As SAConnection
C# public SAConnection Connection { get; set; }

Property value
The default value is a null reference. In Visual Basic it is Nothing.

See also
● “SACommand class” on page 206
● “SACommand members” on page 207

DesignTimeVisible property
Gets or sets a value that indicates if the SACommand should be visible in a Windows Form Designer control. The default is true.

Syntax
Visual Basic
Public Overrides Property DesignTimeVisible As Boolean

C#
public override bool DesignTimeVisible { get; set; }

Property value
True if this SACommand instance should be visible, false if this instance should not be visible. The default is false.

See also
● “SACommand class” on page 206
● “SACommand members” on page 207

Parameters property
A collection of parameters for the current statement. Use question marks in the CommandText to indicate parameters.

Syntax
Visual Basic
Public Readonly Property Parameters As SAParameterCollection

C#
public SAParameterCollection Parameters { get; }
Property value

The parameters of the SQL statement or stored procedure. The default value is an empty collection.

Remarks

When CommandType is set to Text, pass parameters using the question mark placeholder. 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 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

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SAParameterCollection class” on page 396

Transaction property

Specifies the SATransaction object in which the SACommand executes.

Syntax

**Visual Basic**

```vbnet
Public Property Transaction As SATransaction
```

**C#**

```csharp
public SATransaction Transaction { get; set; }
```

Property value

The default value is a null reference. In Visual Basic, this is Nothing.

Remarks

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 SAConnection object as the SACommand object, an exception will be thrown the next time you attempt to execute a statement.

For more information, see “Transaction processing” on page 136.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SATransaction class” on page 439
**UpdatedRowSource property**

Gets or sets how command results are applied to the DataRow when used by the Update method of the SADatasetAdapter.

**Syntax**

Visual Basic

```vbnet
Public Overrides Property UpdatedRowSource As UpdateRowSource
```

C#

```csharp
public override UpdateRowSource UpdatedRowSource { get; set; }
```

**Property value**

One of the UpdatedRowSource values. The default value is UpdateRowSource.OutputParameters. If the command is automatically generated, this property is UpdateRowSource.None.

**Remarks**

UpdatedRowSource.Both, which returns both resultset and output parameters, is not supported.

**See also**

- “SACommand class” on page 206
- “SACommand members” on page 207

**BeginExecuteNonQuery methods**

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.

**BeginExecuteNonQuery() method**

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand.

**Syntax**

Visual Basic

```vbnet
Public Function BeginExecuteNonQuery() As IAsyncResult
```

C#

```csharp
public IAsyncResult BeginExecuteNonQuery();
```

iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
Return value

An 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.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “BeginExecuteNonQuery methods” on page 215
- “EndExecuteNonQuery method” on page 221

BeginExecuteNonQuery(AsyncCallback, Object) method

Initiates the asynchronous execution of a SQL statement or stored procedure that is described by this SACommand, given a callback procedure and state information.

Syntax

Visual Basic

Public Function BeginExecuteNonQuery( _
   ByVal callback As AsyncCallback, _
   ByVal stateObject As Object _) As IAsyncResult

C#

public IAsyncResult BeginExecuteNonQuery( 
   AsyncCallback callback,
   object stateObject
);

Parameters

- callback An 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 IAsyncResult.AsyncState.

Return value

An 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.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “BeginExecuteNonQuery methods” on page 215
- “EndExecuteNonQuery method” on page 221
**BeginExecuteReader methods**

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.

**BeginExecuteReader() 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 database server.

**Syntax**

**Visual Basic**

Public Function BeginExecuteReader() As IAsyncResult

**C#**

public IAsyncResult BeginExecuteReader();

**Return value**

An IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an SDataReader object that can be used to retrieve the returned rows.

**See also**

- “SACommand class” on page 206
- “SACommand members” on page 207
- “BeginExecuteReader methods” on page 217
- “EndExecuteReader method” on page 223
- “SDataReader class” on page 310

**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.

**Syntax**

**Visual Basic**

Public Function BeginExecuteReader( _
    ByVal behavior As CommandBehavior _
) As IAsyncResult

**C#**

public IAsyncResult BeginExecuteReader(
CommandBehavior behavior
);

Parameters

- **behavior** A bitwise combination of CommandBehavior flags describing the results of the query and its effect on the connection.

Return value

An IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an SDataReader object that can be used to retrieve the returned rows.

See also

- “SAClass” on page 206
- “SAClass members” on page 207
- “BeginExecuteReader methods” on page 217
- “EndExecuteReader method” on page 223
- “SDataReader class” on page 310

BeginExecuteReader(AsyncCallback, Object) method

Initiates the asynchronous execution of a SQL statement that is described by the SAClass object, and retrieves the result set, given a callback procedure and state information.

Syntax

Visual Basic

    Public Function BeginExecuteReader( _
            ByVal callback As AsyncCallback, _
            ByVal stateObject As Object _
        ) As IAsyncResult

C#

    public IAsyncResult BeginExecuteReader( _
            AsyncCallback callback, _
            object stateObject _
        );

Parameters

- **callback** An 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 IAsyncResult.AsyncState.
Return value

An IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an SDataReader object that can be used to retrieve the returned rows.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “BeginExecuteReader methods” on page 217
- “EndExecuteReader method” on page 223
- “SDataReader class” on page 310

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.

Syntax

Visual Basic

Public Function BeginExecuteReader( _
    ByVal callback As AsyncCallback, _
    ByVal stateObject As Object, _
    ByVal behavior As CommandBehavior _
) As IAsyncResult

C#

public IAsyncResult BeginExecuteReader( 
    AsyncCallback callback,  
    object stateObject,  
    CommandBehavior behavior  
);

Parameters

- **callback** An 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 IAsyncResult.AsyncState.

- **behavior** A bitwise combination of CommandBehavior flags describing the results of the query and its effect on the connection.

Return value

An IAsyncResult that can be used to poll, wait for results, or both; this value is also needed when invoking EndExecuteReader(IAsyncResult), which returns an SDataReader object that can be used to retrieve the returned rows.
See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “BeginExecuteReader methods” on page 217
- “EndExecuteReader method” on page 223
- “SDataReader class” on page 310

**Cancel method**

Cancels the execution of an SACommand object.

**Syntax**

**Visual Basic**

Public Overrides Sub Cancel()

**C#**

public override void Cancel();

**Remarks**

If there is nothing to cancel, nothing happens. If there is a command in process and the attempt to cancel fails, no exception is generated.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207

**CreateParameter method**

Provides an SAParameter object for supplying parameters to SACommand objects.

**Syntax**

**Visual Basic**

Public Function CreateParameter() As SAParameter

**C#**

public SAParameter CreateParameter();

**Return value**

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

- “SACommand class” on page 206
- “SACommand members” on page 207
- “SAParameter class” on page 382

EndExecuteNonQuery method

Finishes asynchronous execution of a SQL statement or stored procedure.

Syntax

**Visual Basic**

```vbnet
Public Function EndExecuteNonQuery( _
    ByVal asyncResult As IAsyncResult _
) As Integer
```

**C#**

```csharp
public int EndExecuteNonQuery( 
    IAsyncResult asyncResult
);
```

Parameters

- **asyncResult** The IAsyncResult returned by the call to SACommand.BeginExecuteNonQuery.

Return value

The number of rows affected (the same behavior as SACommand.ExecuteNonQuery).

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:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'
    + " WHERE DepartmentID=100",
    conn);
IAAsyncResult res = cmd.BeginExecuteNonQuery();
// perform other work
// this will block until the command completes
int rowCount = cmd.EndExecuteNonQuery(res);
```

(2) Poll the IsCompleted property of the IAsyncResult.

You can poll the IsCompleted property of the IAsyncResult. For example:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'
    + " WHERE DepartmentID=100",
    conn);
IAAsyncResult 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:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand(
    "UPDATE Departments"
    + " SET DepartmentName = 'Engineering'
    + " WHERE DepartmentID=100",
    conn);
IAAsyncResult 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:

```csharp
private void callbackFunction( IAsyncResult ar )
{
    SACommand cmd = (SACommand) ar.AsyncState;
    // this won't block since the command has completed
    int rowCount = cmd.EndExecuteNonQuery();
}
```
private void DoStuff()
{
    SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
    conn.Open();
    SACommand cmd = new SACommand(
        "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
● “SACommand class” on page 206
● “SACommand members” on page 207
● “BeginExecuteNonQuery() method” on page 215

**EndExecuteReader method**

Finishes asynchronous execution of a SQL statement or stored procedure, returning the requested
SADDataReader.

**Syntax**

**Visual Basic**

```vbnet
Public Function EndExecuteReader( _
    ByVal asyncResult As IAsyncResult _
) As SADDataReader
```

**C#**

```csharp
public SADDataReader EndExecuteReader( _
    IAsyncResult asyncResult
);```

**Parameters**

- **asyncResult**  The IAsyncResult returned by the call to SACommand.BeginExecuteReader.

**Return value**

An SADDataReader object that can be used to retrieve the requested rows (the same behavior as
SACommand.ExecuteReader).
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:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
// this will block 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:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
while( !res.IsCompleted ) {
    // do other work
}
// this will 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:

```csharp
SAConnection conn = new SAConnection("DSN=SQL Anywhere 11 Demo");
conn.Open();
SACommand cmd = new SACommand( "SELECT * FROM Departments", conn );
IAsyncResult res = cmd.BeginExecuteReader();
// perform other work
WaitHandle wh = res.AsyncWaitHandle;
wh.WaitOne();
// this will 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:

```csharp
private void callbackFunction( IAsyncResult ar )
{
```
SACommand cmd = (SACommand) ar.AsyncState;
// this won’t block since the command has completed
SADataReader reader = cmd.EndExecuteReader();
}
// elsewhere in the code
private void DoStuff()
{
    SACollection conn = new SACollection("DSN=SQL Anywhere 11 Demo");
    conn.Open();
    SACommand cmd = new SACommand( "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 class” on page 206
● “SACommand members” on page 207
● “BeginExecuteReader() method” on page 217
● “SADataReader class” on page 310

**ExecuteNonQuery method**

Executes a statement that does not return a result set, such as an INSERT, UPDATE, DELETE, or data
definition statement.

**Syntax**

**Visual Basic**

Public Overrides Function ExecuteNonQuery() As Integer

**C#**

public override int ExecuteNonQuery();

**Return value**

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 class” on page 206
- “SACommand members” on page 207
- “ExecuteReader() method” on page 226

**ExecuteReader methods**

Executes a SQL statement that returns a result set.

**ExecuteReader() method**

Executes a SQL statement that returns a result set.

**Syntax**

**Visual Basic**

Public Function ExecuteReader() As SADataReader

**C#**

public SADataReader ExecuteReader();

**Return value**

The result set as an SADataReader object.

**Remarks**

The statement is the current SACommand object, with CommandText and Parameters as needed. The SADataReader object is a read-only, forward-only result set. For modifiable result sets, use an SDataAdapter.

See also

- “SACommand class” on page 206
- “SACommand members” on page 207
- “ExecuteReader methods” on page 226
- “ExecuteNonQuery method” on page 225
- “SADataReader class” on page 310
- “SDataAdapter class” on page 299
- “CommandText property” on page 211
- “Parameters property” on page 213

**ExecuteReader(CommandBehavior) method**

Executes a SQL statement that returns a result set.
Syntax

Visual Basic

Public Function ExecuteReader( _
    ByVal behavior As CommandBehavior _
) As SADataReader

C#

public SADataReader 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.

Return value

The result set as an SADataReader object.

Remarks

The statement is the current SACommand 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

- “SACommand class” on page 206
- “SACommand members” on page 207
- “ExecuteReader methods” on page 226
- “ExecuteNonQuery method” on page 225
- “SADataReader class” on page 310
- “SADataAdapter class” on page 299
- “CommandText property” on page 211
- “Parameters property” on page 213

**ExecuteScalar method**

Executes a statement that returns a single value. If this method is called on a query that returns multiple rows and columns, only the first column of the first row is returned.

Syntax

Visual Basic

Public Overrides Function ExecuteScalar() As Object
C#

public override object ExecuteScalar();

**Return value**

The first column of the first row in the result set, or a null reference if the result set is empty.

**See also**

- “SACommand class” on page 206
- “SACommand members” on page 207

## Prepare method

Prepares or compiles the SACommand on the data source.

**Syntax**

**Visual Basic**

Public Overrides Sub Prepare()

**C#**

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

- “SACommand class” on page 206
- “SACommand members” on page 207
- “ExecuteNonQuery method” on page 225
- “ExecuteReader() method” on page 226
- “ExecuteScalar method” on page 227

**ResetCommandTimeout method**

Resets the CommandTimeout property to its default value of 30 seconds.

**Syntax**

**Visual Basic**

```vbnet
Public Sub ResetCommandTimeout()
```

**C#**

```csharp
public void ResetCommandTimeout();
```

See also

- “SACommand class” on page 206
- “SACommand members” on page 207

**SACommandBuilder class**

A way to generate single-table SQL statements that reconcile changes made to a DataSet with the data in the associated database. This class cannot be inherited.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SACommandBuilder
Inherits DbCommandBuilder
```

**C#**

```csharp
public sealed class SACommandBuilder : DbCommandBuilder
```

See also

- “SACommandBuilder members” on page 230
# SACommandBuilder members

## Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACommandBuilder constructors</td>
<td>Initializes a new instance of the “SACommandBuilder class” on page 229.</td>
</tr>
</tbody>
</table>

## Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CatalogLocation (inherited from DbCommandBuilder)</td>
<td>Sets or gets the CatalogLocation for an instance of the DbCommandBuilder.</td>
</tr>
<tr>
<td>CatalogSeparator (inherited from DbCommandBuilder)</td>
<td>Sets or gets a string used as the catalog separator for an instance of the DbCommandBuilder.</td>
</tr>
<tr>
<td>ConflictOption (inherited from DbCommandBuilder)</td>
<td>Specifies which ConflictOption is to be used by the DbCommandBuilder.</td>
</tr>
<tr>
<td>DataAdapter property</td>
<td>Specifies the SDataAdapter for which to generate statements.</td>
</tr>
<tr>
<td>QuotePrefix (inherited from DbCommandBuilder)</td>
<td>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.</td>
</tr>
<tr>
<td>QuoteSuffix (inherited from DbCommandBuilder)</td>
<td>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.</td>
</tr>
<tr>
<td>SchemaSeparator (inherited from DbCommandBuilder)</td>
<td>Gets or sets the character to be used for the separator between the schema identifier and any other identifiers.</td>
</tr>
<tr>
<td>SetAllValues (inherited from DbCommandBuilder)</td>
<td>Specifies whether all column values in an update statement are included or only changed ones.</td>
</tr>
</tbody>
</table>

## Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeriveParameters method</td>
<td>Populates the Parameters collection of the specified SACommand object. This is used for the stored procedure specified in the SACommand.</td>
</tr>
<tr>
<td>GetDeleteCommand methods</td>
<td>Returns the generated SACommand object that performs DELETE operations on the database when SDataAdapter.Update is called.</td>
</tr>
</tbody>
</table>
### GetInsertCommand methods
Returns the generated SACommand object that performs INSERT operations on the database when an Update is called.

### GetUpdateCommand methods
Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

### QuoteIdentifier method
Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

### RefreshSchema (inherited from DbCommandBuilder)
Clears the commands associated with this `DbCommandBuilder`.

### UnquoteIdentifier method
Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.

---

### See also
- “SACommandBuilder class” on page 229

---

### SACommandBuilder constructors

#### SACommandBuilder() constructor
Initializes an `SACommandBuilder` object.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New()
```

**C#**

```csharp
public SACommandBuilder();
```

### See also
- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
- “SACommandBuilder constructors” on page 231

---

### SACommandBuilder(SDataAdapter) constructor
Initializes an `SACommandBuilder` object.
Syntax

Visual Basic

Public Sub New(_
    ByVal adapter As SADataAdapter _
)

C#

public SACommandBuilder(
    SADataAdapter adapter
);

Parameters

- **adapter** An SADataAdapter object for which to generate reconciliation statements.

See also

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
- “SACommandBuilder constructors” on page 231

**DataAdapter property**

Specifies the SADataAdapter for which to generate statements.

Syntax

Visual Basic

Public Property DataAdapter As SADataAdapter

C#

public SADataAdapter DataAdapter { get; set; }

Property value

An SADataAdapter object.

Remarks

When you create a new instance of SACommandBuilder, any existing SACommandBuilder that is associated with this SADataAdapter is released.

See also

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
DeriveParameters method

Populates the Parameters collection of the specified SACommand object. This is used for the stored procedure specified in the SACommand.

Syntax

Visual Basic

Public Shared Sub DeriveParameters(
    ByVal command As SACommand
)

C#

public static void DeriveParameters(
    SACommand command
);

Parameters

- command  An SACommand object for which to derive parameters.

Remarks

DeriveParameters overwrites any existing parameter information for the SACommand.

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.

See also

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230

GetDeleteCommand methods

Returns the generated SACommand object that performs DELETE operations on the database when SADataAdapter.Update is called.

GetDeleteCommand(Boolean) method

Returns the generated SACommand object that performs DELETE operations on the database when SADataAdapter.Update is called.

Syntax

Visual Basic

Public Function GetDeleteCommand(
    ByVal useColumnsForParameterNames As Boolean
) As SACommand
C#

public SACommand GetDeleteCommand(
    bool useColumnsForParameterNames
);

Parameters

● useColumnsForParameterNames If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Return value

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 SADataset.

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

● “SACommandBuilder class” on page 229
 ● “SACommandBuilder members” on page 230
 ● “GetDeleteCommand methods” on page 233
 ● DbCommandBuilder.RefreshSchema

GetDeleteCommand() method

Returns the generated SACommand object that performs DELETE operations on the database when SADataset.Update is called.

Syntax

Visual Basic

Public Function GetDeleteCommand() As SACommand

C#

public SACommand GetDeleteCommand();

Return value

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 SADDataAdapter.

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

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
- “GetDeleteCommand methods” on page 233
- DbCommandBuilder.RefreshSchema

GetInsertCommand methods

Returns the generated SACommand object that performs INSERT operations on the database when an Update is called.

GetInsertCommand(Boolean) method

Returns the generated SACommand object that performs INSERT operations on the database when an Update is called.

Syntax

Visual Basic

Public Function GetInsertCommand(  _
    ByVal useColumnsForParameterNames As Boolean  _
) As SACommand

C#

public SACommand GetInsertCommand(  
    bool useColumnsForParameterNames
);

Parameters

- useColumnsForParameterNames If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Return value

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 class” on page 229
- “SACommandBuilder members” on page 230
- “GetInsertCommand methods” on page 235
- “GetDeleteCommand() method” on page 234

GetInsertCommand() method

Returns the generated SACommand object that performs INSERT operations on the database when an Update is called.

Syntax

Visual Basic

Public Function GetInsertCommand() As SACommand

C#

public SACommand GetInsertCommand();

Return value

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.
GetUpdateCommand methods

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

GetUpdateCommand(Boolean) method

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

Syntax

Visual Basic

Public Function GetUpdateCommand( _
    ByVal useColumnsForParameterNames As Boolean _
) As SACommand

C#

public SACommand GetUpdateCommand(  
    bool useColumnsForParameterNames  
);  

Parameters

- **useColumnsForParameterNames** If true, generate parameter names matching column names if possible. If false, generate @p1, @p2, and so on.

Return value

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 SADDataAdapter.

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

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
- “GetUpdateCommand methods” on page 237
- DbCommandBuilder.RefreshSchema

GetUpdateCommand() method

Returns the generated SACommand object that performs UPDATE operations on the database when an Update is called.

Syntax

**Visual Basic**

Public Function GetUpdateCommand() As SACommand

**C#**

public SACommand GetUpdateCommand();

Return value

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 SADAdapter.

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

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230
- “GetUpdateCommand methods” on page 237
- DbCommandBuilder.RefreshSchema
**QuotIdentifer method**

Returns the correct quoted form of an unquoted identifier, including properly escaping any embedded quotes in the identifier.

**Syntax**

**Visual Basic**

Public Overrides Function QuotIdentifer(  
  ByVal unquotedIdentifier As String  
) As String

**C#**

public override string QuotIdentifer(  
  string unquotedIdentifier  
);

**Parameters**

- **unquotedIdentifier**  The string representing the unquoted identifier that will have be quoted.

**Return value**

Returns a string representing the quoted form of an unquoted identifier with embedded quotes properly escaped.

**See also**

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230

**UnquotIdentifer method**

Returns the correct unquoted form of a quoted identifier, including properly un-escaping any embedded quotes in the identifier.

**Syntax**

**Visual Basic**

Public Overrides Function UnquotIdentifer(  
  ByVal quotedIdentifier As String  
) As String

**C#**

public override string UnquotIdentifer(  
  string quotedIdentifier  
);

iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
Parameters

- quotedIdentifier The string representing the quoted identifier that will have its embedded quotes removed.

Return value

Returns a string representing the unquoted form of a quoted identifier with embedded quotes properly unescaped.

See also

- “SACommandBuilder class” on page 229
- “SACommandBuilder members” on page 230

SACommLinksOptionsBuilder class

Provides a simple way to create and manage the CommLinks options portion of connection strings used by the SAConnection class. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SACommLinksOptionsBuilder

C#

public sealed class SACommLinksOptionsBuilder

Remarks

The SACommLinksOptionsBuilder 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].

See also

- “SACommLinksOptionsBuilder members” on page 240

SACommLinksOptionsBuilder members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACommLinksOptionsBuilder constructors</td>
<td>Initializes a new instance of the “SACommLinksOptionsBuilder class” on page 240.</td>
</tr>
</tbody>
</table>
Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All property</td>
<td>Gets or sets the ALL CommLinks option.</td>
</tr>
<tr>
<td>ConnectionString property</td>
<td>Gets or sets the connection string being built.</td>
</tr>
<tr>
<td>SharedMemory property</td>
<td>Gets or sets the SharedMemory protocol.</td>
</tr>
<tr>
<td>TcpOptionsBuilder property</td>
<td>Gets or sets a TcpOptionsBuilder object used to create a TCP options string.</td>
</tr>
<tr>
<td>TcpOptionsString property</td>
<td>Gets or sets a string of TCP options.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUseLongNameAsKeyword method</td>
<td>Gets a boolean values that indicates whether long connection parameter names are used in the connection string.</td>
</tr>
<tr>
<td>SetUseLongNameAsKeyword method</td>
<td>Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.</td>
</tr>
<tr>
<td>ToString method</td>
<td>Converts the SACommLinksOptionsBuilder object to a string representation.</td>
</tr>
</tbody>
</table>

See also

- “SACommLinksOptionsBuilder class” on page 240

**SACommLinksOptionsBuilder constructors**

Initializes a new instance of the “SACommLinksOptionsBuilder class” on page 240.

**SACommLinksOptionsBuilder() constructor**

Initializes an SACommLinksOptionsBuilder object.

**Syntax**

**Visual Basic**

```visualbasic
Public Sub New()
```
C#

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( );
```

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240
- “SACommLinksOptionsBuilder constructors” on page 241

SACommLinksOptionsBuilder(String) constructor

Initializes an SACommLinksOptionsBuilder object.

Syntax

Visual Basic

Public Sub New(_
    ByVal options As String _
)

C#

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("TCPIP(DoBroadcast=ALL;Timeout=20)" );
```
See also
  ● “SACommLinksOptionsBuilder class” on page 240
  ● “SACommLinksOptionsBuilder members” on page 240
  ● “SACommLinksOptionsBuilder constructors” on page 241

All property

Gets or sets the ALL CommLinks option.

Syntax
  Visual Basic
       Public Property All As Boolean
  C#
       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.

See also
  ● “SACommLinksOptionsBuilder class” on page 240
  ● “SACommLinksOptionsBuilder members” on page 240

ConnectionString property

Gets or sets the connection string being built.

Syntax
  Visual Basic
       Public Property ConnectionString As String
  C#
       public string ConnectionString { get; set; }

Remarks
   The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.
See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240

**SharedMemory property**

Gets or sets the SharedMemory protocol.

**Syntax**

**Visual Basic**

Public Property **SharedMemory** As Boolean

**C#**

public bool **SharedMemory** { get; set; }

**Remarks**

The SACommLinksOptionsBuilder class is not available in the .NET Compact Framework 2.0.

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240

**TcpOptionsBuilder property**

Gets or sets a TcpOptionsBuilder object used to create a TCP options string.

**Syntax**

**Visual Basic**

Public Property **TcpOptionsBuilder** As SATcpOptionsBuilder

**C#**

public SATcpOptionsBuilder **TcpOptionsBuilder** { get; set; }

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240

**TcpOptionsString property**

Gets or sets a string of TCP options.
Syntax

Visual Basic
Public Property **TcpOptionsString** As String

C#
public string **TcpOptionsString** { get; set; }

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240

---

### GetUseLongNameAsKeyword method

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

**Syntax**

**Visual Basic**

Public Function **GetUseLongNameAsKeyword**() As Boolean

**C#**

public bool **GetUseLongNameAsKeyword();**

**Return value**

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 class” on page 240
- “SACommLinksOptionsBuilder members” on page 240
- “SetUseLongNameAsKeyword method” on page 245

---

### SetUseLongNameAsKeyword method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.
Syntax

Visual Basic

Public Sub SetUseLongNameAsKeyword( 
    ByVal useLongNameAsKeyword As Boolean 
)

C#

public void SetUseLongNameAsKeyword( 
    bool useLongNameAsKeyword 
);

Parameters

- **useLongNameAsKeyword**  
  A boolean value that indicates whether the long connection parameter name is used in the connection string.

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240
- “GetUseLongNameAsKeyword method” on page 245

**ToString method**

Converts the SACommLinksOptionsBuilder object to a string representation.

Syntax

Visual Basic

Public Overrides Function ToString() As String

C#

public override string ToString();

Return value

The options string being built.

See also

- “SACommLinksOptionsBuilder class” on page 240
- “SACommLinksOptionsBuilder members” on page 240

**SAConnection class**

Represents a connection to a SQL Anywhere database. This class cannot be inherited.
Syntax

Visual Basic

Public NotInheritable Class SAConnection
Inherits DbConnection

C#

public sealed class SAConnection : DbConnection

Remarks

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

See also

● “SAConnection members” on page 247

SAConnection members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAConnection constructors</td>
<td>Initializes a new instance of the “SAConnection class” on page 246.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionString property</td>
<td>Provides the database connection string.</td>
</tr>
<tr>
<td>ConnectionTimeout property</td>
<td>Gets the number of seconds before a connection attempt times out with an error.</td>
</tr>
<tr>
<td>DataSource property</td>
<td>Gets the name of the database server.</td>
</tr>
<tr>
<td>Database property</td>
<td>Gets the name of the current database.</td>
</tr>
<tr>
<td>InitString property</td>
<td>A command that is executed immediately after the connection is established.</td>
</tr>
<tr>
<td>ServerVersion property</td>
<td>Gets a string that contains the version of the instance of SQL Anywhere to which the client is connected.</td>
</tr>
<tr>
<td>State property</td>
<td>Indicates the state of the SAConnection object.</td>
</tr>
</tbody>
</table>
Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginTransaction methods</td>
<td>Returns a transaction object. 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.</td>
</tr>
<tr>
<td>ChangeDatabase method</td>
<td>Changes the current database for an open SAConnection.</td>
</tr>
<tr>
<td>ChangePassword method</td>
<td>Changes the password for the user indicated in the connection string to the supplied new password.</td>
</tr>
<tr>
<td>ClearAllPools method</td>
<td>Empties all connection pools.</td>
</tr>
<tr>
<td>ClearPool method</td>
<td>Empties the connection pool associated with the specified connection.</td>
</tr>
<tr>
<td>Close method</td>
<td>Closes a database connection.</td>
</tr>
<tr>
<td>CreateCommand method</td>
<td>Initializes an SACommand object.</td>
</tr>
<tr>
<td>EnlistDistributedTransaction method</td>
<td>Enlists in the specified transaction as a distributed transaction.</td>
</tr>
<tr>
<td>EnlistTransaction method</td>
<td>Enlists in the specified transaction as a distributed transaction.</td>
</tr>
<tr>
<td>GetSchema methods</td>
<td>Returns the list of supported schema collections.</td>
</tr>
<tr>
<td>Open method</td>
<td>Opens a database connection with the property settings specified by the SAConnection.ConnectionString.</td>
</tr>
</tbody>
</table>

Public events

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoMessage event</td>
<td>Occurs when the SQL Anywhere database server returns a warning or informational message.</td>
</tr>
<tr>
<td>StateChange event</td>
<td>Occurs when the state of the SAConnection object changes.</td>
</tr>
</tbody>
</table>

See also
- “SAConnection class” on page 246

SAConnection constructors

Initializes a new instance of the “SAConnection class” on page 246.
SAConnection() constructor

Initializes an SAConnection object. The connection must be opened before you can perform any operations against the database.

Syntax
Visual Basic
Public Sub New()

C#
public SAConnection();

See also
- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “SAConnection constructors” on page 248

SAConnection(String) constructor

Initializes an SAConnection object. The connection must then be opened before you can perform any operations against the database.

Syntax
Visual Basic
Public Sub New(  ByVal connectionString As String  )

C#
public SAConnection(  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].

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;ENG=hr;DBN=policies");
conn.Open();

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “SAConnection constructors” on page 248
- “SAConnection class” on page 246

**ConnectionString property**

Provides the database connection string.

**Syntax**

**Visual Basic**

Public Overrides Property ConnectionString As String

**C#**

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.

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. SAConnection 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"s". 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).

Example
The following statements set a connection string for an ODBC data source named SQL Anywhere 11 Demo and open the connection.

```csharp
SAConnection conn = new SAConnection();
conn.ConnectionString = "DSN=SQL Anywhere 11 Demo";
conn.Open();
```

See also
- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “SAConnection class” on page 246
- “Open method” on page 267

ConnectionTimeout property
Gets the number of seconds before a connection attempt times out with an error.

Syntax
Visual Basic
```
Public Overrides Readonly Property ConnectionTimeout As Integer
```

C#
```
public override int ConnectionTimeout { get; }
```

Property value
15 seconds

Example
The following statement displays the value of the ConnectionTimeout.
```
MessageBox.Show( conn.ConnectionTimeout.ToString( ) );
```

See also
- “SAConnection class” on page 246
- “SAConnection members” on page 247
**DataSource property**

Gets the name of the database server.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property DataSource As String

**C#**

public override string DataSource { get; }

**Remarks**

If the connection is opened, the SAConnection object returns the ServerName server property. Otherwise, the SAConnection object looks in the connection string in the following order: EngineName, ServerName, ENG.

**See also**

- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “SAConnection class” on page 246

**Database property**

Gets the name of the current database.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property Database As String

**C#**

custom public override string Database { get; }

**Remarks**

If the connection is opened, SAConnection returns the name of the current database. Otherwise, SAConnection looks in the connection string in the following order: DatabaseName, DBN, DataSourceName, DataSource, DSN, DatabaseFile, DBF.

**See also**

- “SAConnection class” on page 246
- “SAConnection members” on page 247
InitString property

A command that is executed immediately after the connection is established.

Syntax

Visual Basic

Public Property InitString As String

C#

public string InitString { get; set; }

Remarks

The InitString will be executed immediately after the connection is opened.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247

ServerVersion property

Gets a string that contains the version of the instance of SQL Anywhere to which the client is connected.

Syntax

Visual Basic

Public Overrides Readonly Property ServerVersion As String

C#

public override string ServerVersion { get; }

Property value

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.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247
**State property**

Indicates the state of the SAConnection object.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property State As ConnectionState

**C#**

public override ConnectionState State { get; }

**Property value**

A ConnectionState enumeration.

**See also**

- “SAConnection class” on page 246
- “SAConnection members” on page 247

**BeginTransaction methods**

Returns a transaction object. 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.

**BeginTransaction() method**

Returns a transaction object. 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.

**Syntax**

**Visual Basic**

Public Function BeginTransaction() As SATransaction

**C#**

public SATransaction BeginTransaction();

**Return value**

An SATransaction object representing the new transaction.

**Remarks**

To associate a command with a transaction object, use the SACommand.Transaction property.
BeginTransaction(IsolationLevel) method

Returns a transaction object. 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.

Syntax

Visual Basic

Public Function BeginTransaction( _
    ByVal isolationLevel As IsolationLevel _
) As SATransaction

C#

public SATransaction BeginTransaction( _
    IsolationLevel isolationLevel _
);

Parameters

- isolationLevel: A member of the SAIsolationLevel enumeration. The default value is ReadCommitted.

Return value

An SATransaction object representing the new transaction.

Remarks

To associate a command with a transaction object, use the SACommand.Transaction property.

Example

SATransaction tx = conn.BeginTransaction( _
    SAIsolationLevel.ReadUncommitted );

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “BeginTransaction methods” on page 254
- “SATransaction class” on page 439
- “Transaction property” on page 214
- “SAIsolationLevel enumeration” on page 370
BeginTransaction(SAIsolationLevel) method

Returns a transaction object. 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.

Syntax

Visual Basic

Public Function BeginTransaction(  
    ByVal isolationLevel As SAIsolationLevel  
) As SATransaction

C#

public SATransaction BeginTransaction(  
    SAIsolationLevel isolationLevel  
);

Parameters

● isolationLevel  A member of the SAIsolationLevel enumeration. The default value is ReadCommitted.

Return value

An SATransaction object representing the new transaction.

For more information, see “Transaction processing” on page 136.

For more information, see “Typical types of inconsistency” [SQL Anywhere Server - SQL Usage].

Remarks

To associate a command with a transaction object, use the SACommand.Transaction property.

See also

● “SAConnection class” on page 246
● “SAConnection members” on page 247
● “BeginTransaction methods” on page 254
● “SATransaction class” on page 439
● “Transaction property” on page 214
● “SAIsolationLevel enumeration” on page 370
● “Commit method” on page 443
● “Rollback() method” on page 443
● “Rollback(String) method” on page 444

ChangeDatabase method

Changes the current database for an open SAConnection.
Syntax

Visual Basic

Public Overrides Sub ChangeDatabase(ByVal database As String)

C#

public override void ChangeDatabase(string database);

Parameters

- **database**  The name of the database to use instead of the current database.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247

---

**ChangePassword method**

Changes the password for the user indicated in the connection string to the supplied new password.

Syntax

Visual Basic

Public Shared Sub ChangePassword(ByVal connectionString As String, ByVal newPassword As String)

C#

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.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247
ClearAllPools method

Empties all connection pools.

Syntax

Visual Basic

Public Shared Sub ClearAllPools()

C#

public static void ClearAllPools();

See also

● “SAConnection class” on page 246
● “SAConnection members” on page 247

ClearPool method

Empties the connection pool associated with the specified connection.

Syntax

Visual Basic

Public Shared Sub ClearPool(
  ByVal connection As SAConnection
)

C#

public static void ClearPool(
  SAConnection connection
)

Parameters

● connection The SAConnection object to be cleared from the pool.

See also

● “SAConnection class” on page 246
● “SAConnection members” on page 247
● “SAConnection class” on page 246

Close method

Closes a database connection.
Syntax

Visual Basic

Public Overrides Sub Close()

C#

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.

See also

● “SAConnection class” on page 246
● “SAConnection members” on page 247

CreateCommand method

Initializes an SACommand object.

Syntax

Visual Basic

Public Function CreateCommand() As SACommand

C#

public SACommand CreateCommand();

Return value

An SACommand object.

Remarks

The command object is associated with the SAConnection object.

See also

● “SAConnection class” on page 246
● “SAConnection members” on page 247
● “SACommand class” on page 206
● “SAConnection class” on page 246

EnlistDistributedTransaction method

Enlists in the specified transaction as a distributed transaction.
EnlistDistributedTransaction method

Enlists in the specified transaction as a distributed transaction.

Syntax

Visual Basic
Public Sub EnlistDistributedTransaction( _
    ByVal transaction As ITransaction _
)

C#
public void EnlistDistributedTransaction(
    ITransaction transaction
);

Parameters

- transaction   A reference to an existing System.EnterpriseServices.ITransaction in which to enlist.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247

EnlistTransaction method

Enlists in the specified transaction as a distributed transaction.

Syntax

Visual Basic
Public Overrides Sub EnlistTransaction( _
    ByVal transaction As Transaction _
)

C#
public override void EnlistTransaction(
    Transaction transaction
);

Parameters

- transaction   A reference to an existing System.Transactions.Transaction in which to enlist.

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247

GetSchema methods

Returns the list of supported schema collections.
GetSchema() method

Returns the list of supported schema collections.

Syntax

Visual Basic
Public Overrides Function GetSchema() As DataTable

C#
public override DataTable GetSchema();

Remarks
See GetSchema(string,string[]) for a description of the available metadata.

See also
- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “GetSchema methods” on page 260
- “GetSchema(String, String[]) method” on page 262

GetSchema(String) method

Returns information for the specified metadata collection for this SAConnection object.

Syntax

Visual Basic
Public Overrides Function GetSchema( _
    ByVal collection As String _
) As DataTable

C#
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.
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.

Syntax

Visual Basic

Public Overrides Function GetSchema( _
    ByVal collection As String, _
    ByVal restrictions As String() _) _
    As DataTable

C#

public override DataTable GetSchema(  
    string collection,  
    string[] restrictions  
);  

Return value

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).

<table>
<thead>
<tr>
<th>Collection</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Returns information on all columns in the database.</td>
</tr>
<tr>
<td>DataSourceInformation</td>
<td>Returns information about the database server.</td>
</tr>
<tr>
<td>DataTypes</td>
<td>Returns a list of supported data types.</td>
</tr>
</tbody>
</table>
### Collection Metadata

<table>
<thead>
<tr>
<th>Collection</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForeignKeys</td>
<td>Returns information on all foreign keys in the database.</td>
</tr>
<tr>
<td>IndexColumns</td>
<td>Returns information on all index columns in the database.</td>
</tr>
<tr>
<td>Indexes</td>
<td>Returns information on all indexes in the database.</td>
</tr>
<tr>
<td>MetaDataCollections</td>
<td>Returns a list of all collection names.</td>
</tr>
<tr>
<td>ProcedureParameters</td>
<td>Returns information on all procedure parameters in the database.</td>
</tr>
<tr>
<td>Procedures</td>
<td>Returns information on all procedures in the database.</td>
</tr>
<tr>
<td>ReservedWords</td>
<td>Returns a list of reserved words used by SQL Anywhere.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Returns information on restrictions used in GetSchema.</td>
</tr>
<tr>
<td>Tables</td>
<td>Returns information on all tables in the database.</td>
</tr>
<tr>
<td>UserDefinedTypes</td>
<td>Returns information on all user-defined data types in the database.</td>
</tr>
<tr>
<td>Users</td>
<td>Returns information on all users in the database.</td>
</tr>
<tr>
<td>ViewColumns</td>
<td>Returns information on all columns in views in the database.</td>
</tr>
<tr>
<td>Views</td>
<td>Returns information on all views in the database.</td>
</tr>
</tbody>
</table>

These collection names are also available as read-only properties in the SAMetaDataCollectionNames class.

The restrictions 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:

```csharp
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:

```csharp
GetSchema( "Tables", new string[ ] { NULL, "my_table", NULL } )
```

This returns information on all tables named my_table.

```csharp
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)

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” on page 246
- “SAConnection members” on page 247
- “GetSchema methods” on page 260
- “SAConnection class” on page 246

Open method

Opens a database connection with the property settings specified by the SAConnection.ConnectionString.

Syntax

Visual Basic

Public Overrides Sub Open()

C#

public override void Open();

See also

- “SAConnection class” on page 246
- “SAConnection members” on page 247
- “ConnectionString property” on page 250

InfoMessage event

Occurs when the SQL Anywhere database server returns a warning or informational message.

Syntax

Visual Basic

Public Event InfoMessage As SAInfoMessageEventHandler

C#

public event SAInfoMessageEventHandler InfoMessage ;
Remarks
The event handler receives an argument of type SaInfoMessageEventArgs containing data related to this event. The following SAaInfoMessageEventArgs 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.

Event data
- **MessageType** Returns the type of the message. This can be one of: Action, Info, Status, or Warning.
- **Errors** Returns the collection of messages sent from the data source.
- **Message** Returns the full text of the error sent from the data source.
- **Source** Returns the name of the SQL Anywhere .NET Data Provider.
- **NativeError** Returns the SQL code returned by the database.

See also
- “SAConnection class” on page 246
- “SAConnection members” on page 247

**StateChange event**

Occurs when the state of the SAConnection object changes.

Syntax

**Visual Basic**

Public Overrides Event StateChange As StateChangeEventHandler

**C#**

cpublic 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.

Event data
- **CurrentState** Gets the new state of the connection. The connection object will be in the new state already when the event is fired.
- **OriginalState** Gets the original state of the connection.
**SAConnectionStringBuilder class**

Provides a simple way to create and manage the contents of connection strings used by the SAConnection class. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SAConnectionStringBuilder  
    Inherits SAConnectionStringBuilderBase

**C#**

public sealed class SAConnectionStringBuilder : SAConnectionStringBuilderBase

**Remarks**

The SAConnectionStringBuilder class inherits SAConnectionStringBuilderBase, which inherits DbConnectionStringBuilder.

**Restrictions**: The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.

**Inherits**: “SAConnectionStringBuilderBase class” on page 291

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

**See also**

- “SAConnectionStringBuilder members” on page 269

---

### SAConnectionStringBuilder members

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAConnectionStringBuilder constructors</td>
<td>Initializes a new instance of the “SAConnectionStringBuilder class” on page 269.</td>
</tr>
</tbody>
</table>
### Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppInfo property</td>
<td>Gets or sets the AppInfo connection property.</td>
</tr>
<tr>
<td>AutoStart property</td>
<td>Gets or sets the AutoStart connection property.</td>
</tr>
<tr>
<td>AutoStop property</td>
<td>Gets or sets the AutoStop connection property.</td>
</tr>
<tr>
<td><strong>BrowsableConnectionString</strong> (inherited from DbConnectionStringStringBuilder)**</td>
<td>Gets or sets a value that indicates whether the DbConnectionStringStringBuilder.ConnectionString is visible in Visual Studio designers.</td>
</tr>
<tr>
<td>Charset property</td>
<td>Gets or sets the Charset connection property.</td>
</tr>
<tr>
<td>CommBufferSize property</td>
<td>Gets or sets the CommBufferSize connection property.</td>
</tr>
<tr>
<td>CommLinks property</td>
<td>Gets or sets the CommLinks property.</td>
</tr>
<tr>
<td>Compress property</td>
<td>Gets or sets the Compress connection property.</td>
</tr>
<tr>
<td>CompressionThreshold property</td>
<td>Gets or sets the CompressionThreshold connection property.</td>
</tr>
<tr>
<td>ConnectionLifetime property</td>
<td>Gets or sets the ConnectionLifetime connection property.</td>
</tr>
<tr>
<td>ConnectionName property</td>
<td>Gets or sets the ConnectionName connection property.</td>
</tr>
<tr>
<td>ConnectionReset property</td>
<td>Gets or sets the ConnectionReset connection property.</td>
</tr>
<tr>
<td><strong>ConnectionString</strong> (inherited from DbConnectionStringStringBuilder)**</td>
<td>Gets or sets the connection string associated with the DbConnectionStringStringBuilder.</td>
</tr>
<tr>
<td>ConnectionTimeout property</td>
<td>Gets or sets the ConnectionTimeout connection property.</td>
</tr>
<tr>
<td>Count (inherited from DbConnectionStringStringBuilder)**</td>
<td>Gets the current number of keys that are contained within the DbConnectionStringStringBuilder.ConnectionString.</td>
</tr>
<tr>
<td>DataSourceName property</td>
<td>Gets or sets the DataSourceName connection property.</td>
</tr>
<tr>
<td>DatabaseFile property</td>
<td>Gets or sets the DatabaseFile connection property.</td>
</tr>
<tr>
<td>DatabaseKey property</td>
<td>Gets or sets the DatabaseKey connection property.</td>
</tr>
<tr>
<td>DatabaseName property</td>
<td>Gets or sets the DatabaseName connection property.</td>
</tr>
<tr>
<td>DatabaseSwitches property</td>
<td>Gets or sets the DatabaseSwitches connection property.</td>
</tr>
<tr>
<td>DisableMultiRowFetch property</td>
<td>Gets or sets the DisableMultiRowFetch connection property.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Elevate property</td>
<td>Gets or sets the Elevate connection property.</td>
</tr>
<tr>
<td>EncryptedPassword property</td>
<td>Gets or sets the EncryptedPassword connection property.</td>
</tr>
<tr>
<td>Encryption property</td>
<td>Gets or sets the Encryption connection property.</td>
</tr>
<tr>
<td>Enlist property</td>
<td>Gets or sets the Enlist connection property.</td>
</tr>
<tr>
<td>FileDataSourceName property</td>
<td>Gets or sets the FileDataSourceName connection property.</td>
</tr>
<tr>
<td>ForceStart property</td>
<td>Gets or sets the ForceStart connection property.</td>
</tr>
<tr>
<td>IdleTimeout property</td>
<td>Gets or sets the IdleTimeout connection property.</td>
</tr>
<tr>
<td>Integrated property</td>
<td>Gets or sets the Integrated connection property.</td>
</tr>
<tr>
<td>IsFixedSize property</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder has a fixed size.</td>
</tr>
<tr>
<td>IsReadOnly property</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder is read-only.</td>
</tr>
<tr>
<td>Item property</td>
<td>Gets or sets the value of the connection keyword.</td>
</tr>
<tr>
<td>Kerberos property</td>
<td>Gets or sets the Kerberos connection property.</td>
</tr>
<tr>
<td>Keys property</td>
<td>Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.</td>
</tr>
<tr>
<td>Language property</td>
<td>Gets or sets the Language connection property.</td>
</tr>
<tr>
<td>LazyClose property</td>
<td>Gets or sets the LazyClose connection property.</td>
</tr>
<tr>
<td>LivenessTimeout property</td>
<td>Gets or sets the LivenessTimeout connection property.</td>
</tr>
<tr>
<td>LogFile property</td>
<td>Gets or sets the LogFile connection property.</td>
</tr>
<tr>
<td>MaxPoolSize property</td>
<td>Gets or sets the MaxPoolSize connection property.</td>
</tr>
<tr>
<td>MinPoolSize property</td>
<td>Gets or sets the MinPoolSize connection property.</td>
</tr>
<tr>
<td>NewPassword property</td>
<td>Gets or sets the NewPassword connection property.</td>
</tr>
<tr>
<td>Password property</td>
<td>Gets or sets the Password connection property.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PersistSecurityInfo property</td>
<td>Gets or sets the PersistSecurityInfo connection property.</td>
</tr>
<tr>
<td>Pooling property</td>
<td>Gets or sets the Pooling connection property.</td>
</tr>
<tr>
<td>PrefetchBuffer property</td>
<td>Gets or sets the PrefetchBuffer connection property.</td>
</tr>
<tr>
<td>PrefetchRows property</td>
<td>Gets or sets the PrefetchRows connection property. The default value is 200.</td>
</tr>
<tr>
<td>RetryConnectionTimeout property</td>
<td>Gets or sets the RetryConnectionTimeout property.</td>
</tr>
<tr>
<td>ServerName property</td>
<td>Gets or sets the ServerName connection property.</td>
</tr>
<tr>
<td>StartLine property</td>
<td>Gets or sets the StartLine connection property.</td>
</tr>
<tr>
<td>Unconditional property</td>
<td>Gets or sets the Unconditional connection property.</td>
</tr>
<tr>
<td>UserID property</td>
<td>Gets or sets the UserID connection property.</td>
</tr>
<tr>
<td>Values (inherited from DbConnectionStringBuilder)</td>
<td>Gets an ICollection that contains the values in the DbConnectionStringBuilder.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (inherited from DbConnectionStringBuilder)</td>
<td>Adds an entry with the specified key and value into the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td>Clear (inherited from DbConnectionStringBuilder)</td>
<td>Clears the contents of the DbConnectionStringBuilder instance.</td>
</tr>
<tr>
<td>ContainsKey method (inherited from SAConnectionStringBuilderBase)</td>
<td>Determines whether the SAConnectionStringBuilder object contains a specific keyword.</td>
</tr>
<tr>
<td>EquivalentTo (inherited from DbConnectionStringBuilder)</td>
<td>Compares the connection information in this DbConnectionStringBuilder object with the connection information in the supplied object.</td>
</tr>
<tr>
<td>GetKeyword method (inherited from SAConnectionStringBuilderBase)</td>
<td>Gets the keyword for specified SAConnectionStringBuilder property.</td>
</tr>
<tr>
<td>GetUseLongNameAsKeyword method (inherited from SAConnectionStringBuilderBase)</td>
<td>Gets a boolean values that indicates whether long connection parameter names are used in the connection string.</td>
</tr>
</tbody>
</table>
**Member name** | **Description**
--- | ---
Remove method (inherited from SAConnectionStringBuilder-Base) | Removes the entry with the specified key from the SAConnectionStringBuilder instance.

SetUseLongNameAsKeyword method (inherited from SAConnectionStringBuilderBase) | Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.

ShouldSerialize method (inherited from SAConnectionStringBuilderBase) | Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

ToString (inherited from DbConnectionStringBuilder) | Returns the connection string associated with this DbConnectionStringBuilder.

TryGetValue method (inherited from SAConnectionStringBuilderBase) | Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

---

**See also**

- “SAConnectionStringBuilder class” on page 269

---

**SAConnectionStringBuilder constructors**

Initializes a new instance of the “SAConnectionStringBuilder class” on page 269.

---

**SAConnectionStringBuilder() constructor**

Initializes a new instance of the SAConnectionStringBuilder class.

**Syntax**

Visual Basic

Public Sub New()

C#

public SAConnectionStringBuilder();

**Remarks**

*Restrictions:* The SAConnectionStringBuilder class is not available in the .NET Compact Framework 2.0.
SAConnectionStringBuilder(String) constructor

Initializes a new instance of the SAConnectionStringBuilder class.

Syntax

Visual Basic

Public Sub New(
   ByVal connectionString As String
)

C#

class SAConnectionStringBuilder{
    public SAConnectionStringBuilder(  
        string connectionString  
    );
}

Parameters

- **connectionString**  The basis for the object’s internal connection information.Parsed into key=value pairs.
  
  For a list of connection parameters, see “Connection parameters” [*SQL Anywhere Server - Database Administration*].

Remarks

**Restrictions:** 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;ENG=hr;DBN=policies");
```

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269
- “SAConnectionStringBuilder constructors” on page 273
**AppInfo property**

Gets or sets the AppInfo connection property.

**Syntax**

**Visual Basic**

Public Property AppInfo As String

**C#**

public string AppInfo { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**AutoStart property**

Gets or sets the AutoStart connection property.

**Syntax**

**Visual Basic**

Public Property AutoStart As String

**C#**

public string AutoStart { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**AutoStop property**

Gets or sets the AutoStop connection property.

**Syntax**

**Visual Basic**

Public Property AutoStop As String

**C#**

public string AutoStop { get; set; }
See also
  ● “SAConnectionStringBuilder class” on page 269
  ● “SAConnectionStringBuilder members” on page 269

## Charset property

Gets or sets the Charset connection property.

### Syntax

**Visual Basic**

Public Property Charset As String

**C#**

```csharp
public string Charset { get; set; }
```

See also
  ● “SAConnectionStringBuilder class” on page 269
  ● “SAConnectionStringBuilder members” on page 269

## CommBufferSize property

Gets or sets the CommBufferSize connection property.

### Syntax

**Visual Basic**

Public Property CommBufferSize As Integer

**C#**

```csharp
public int CommBufferSize { get; set; }
```

See also
  ● “SAConnectionStringBuilder class” on page 269
  ● “SAConnectionStringBuilder members” on page 269

## CommLinks property

 Gets or sets the CommLinks property.
Syntax

Visual Basic

Public Property CommLinks As String

C#

public string CommLinks { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

Compress property

Gets or sets the Compress connection property.

Syntax

Visual Basic

Public Property Compress As String

C#

public string Compress { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

CompressionThreshold property

Gets or sets the CompressionThreshold connection property.

Syntax

Visual Basic

Public Property CompressionThreshold As Integer

C#

public int CompressionThreshold { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269
ConnectionLifetime property

Gets or sets the ConnectionLifetime connection property.

Syntax

Visual Basic

Public Property ConnectionLifetime As Integer

C#

public int ConnectionLifetime { get; set; }

See also

● “SACConnectionStringBuilder class” on page 269
● “SACConnectionStringBuilder members” on page 269

ConnectionName property

Gets or sets the ConnectionName connection property.

Syntax

Visual Basic

Public Property ConnectionName As String

C#

public string ConnectionName { get; set; }

See also

● “SACConnectionStringBuilder class” on page 269
● “SACConnectionStringBuilder members” on page 269

ConnectionReset property

Gets or sets the ConnectionReset connection property.

Syntax

Visual Basic

Public Property ConnectionReset As Boolean

C#

public bool ConnectionReset { get; set; }
Property value
A DataTable that contains schema information.

See also
● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

ConnectionTimeout property
Gets or sets the ConnectionTimeout connection property.

Syntax
Visual Basic
Public Property ConnectionTimeout As Integer

C#
public int ConnectionTimeout { get; set; }

Example
The following statement displays the value of the ConnectionTimeout property.
MessageBox.Show( connString.ConnectionTimeout.ToString() );

See also
● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

DataSourceName property
Gets or sets the DataSourceName connection property.

Syntax
Visual Basic
Public Property DataSourceName As String

C#
public string DataSourceName { get; set; }

See also
● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269
**DatabaseFile property**

Gets or sets the DatabaseFile connection property.

**Syntax**

**Visual Basic**

Public Property DatabaseFile As String

**C#**

public string DatabaseFile { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**DatabaseKey property**

Gets or sets the DatabaseKey connection property.

**Syntax**

**Visual Basic**

Public Property DatabaseKey As String

**C#**

public string DatabaseKey { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**DatabaseName property**

Gets or sets the DatabaseName connection property.

**Syntax**

**Visual Basic**

Public Property DatabaseName As String

**C#**

public string DatabaseName { get; set; }
See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**DatabaseSwitches property**

Gets or sets the DatabaseSwitches connection property.

**Syntax**

**Visual Basic**

Public Property DatabaseSwitches As String

**C#**

public string DatabaseSwitches { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**DisableMultiRowFetch property**

Gets or sets the DisableMultiRowFetch connection property.

**Syntax**

**Visual Basic**

Public Property DisableMultiRowFetch As String

**C#**

public string DisableMultiRowFetch { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**Elevate property**

Gets or sets the Elevate connection property.
Syntax

Visual Basic

Public Property Elevate As String

C#

public string Elevate { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

EncryptedPassword property

Gets or sets the EncryptedPassword connection property.

Syntax

Visual Basic

Public Property EncryptedPassword As String

C#

public string EncryptedPassword { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

Encryption property

Gets or sets the Encryption connection property.

Syntax

Visual Basic

Public Property Encryption As String

C#

public string Encryption { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269
Enlist property

Gets or sets the Enlist connection property.

Syntax

Visual Basic

Public Property Enlist As Boolean

C#

public bool Enlist { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

FileDataSourceName property

Gets or sets the FileDataSourceName connection property.

Syntax

Visual Basic

Public Property FileDataSourceName As String

C#

public string FileDataSourceName { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

ForceStart property

Gets or sets the ForceStart connection property.

Syntax

Visual Basic

Public Property ForceStart As String

C#

public string ForceStart { get; set; }
See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

IdleTimeout property

Gets or sets the IdleTimeout connection property.

Syntax

Visual Basic

Public Property IdleTimeout As Integer

C#

public int IdleTimeout { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

Integrated property

Gets or sets the Integrated connection property.

Syntax

Visual Basic

Public Property Integrated As String

C#

public string Integrated { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

Kerberos property

Gets or sets the Kerberos connection property.
Syntax

**Visual Basic**

Public Property **Kerberos** As String

**C#**

```csharp
public string Kerberos { get; set; }
```

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

### Language property

Gets or sets the Language connection property.

**Syntax**

**Visual Basic**

Public Property **Language** As String

**C#**

```csharp
public string Language { get; set; }
```

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

### LazyClose property

Gets or sets the LazyClose connection property.

**Syntax**

**Visual Basic**

Public Property **LazyClose** As String

**C#**

```csharp
public string LazyClose { get; set; }
```

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269
**LivenessTimeout property**

Gets or sets the LivenessTimeout connection property.

**Syntax**

**Visual Basic**

Public Property LivenessTimeout As Integer

**C#**

```csharp
public int LivenessTimeout { get; set; }
```

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**LogFile property**

Gets or sets the LogFile connection property.

**Syntax**

**Visual Basic**

Public Property LogFile As String

**C#**

```csharp
public string LogFile { get; set; }
```

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**MaxPoolSize property**

Gets or sets the MaxPoolSize connection property.

**Syntax**

**Visual Basic**

Public Property MaxPoolSize As Integer

**C#**

```csharp
public int MaxPoolSize { get; set; }
```
MinPoolSize property

Gets or sets the MinPoolSize connection property.

Syntax

Visual Basic

Public Property MinPoolSize As Integer

C#

public int MinPoolSize { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

NewPassword property

Gets or sets the NewPassword connection property.

Syntax

Visual Basic

Public Property NewPassword As String

C#

public string NewPassword { get; set; }

See also

● “SAConnectionStringBuilder class” on page 269
● “SAConnectionStringBuilder members” on page 269

Password property

Gets or sets the Password connection property.
Syntax

**Visual Basic**

Public Property **Password** As String

**C#**

public string **Password** { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**PersistSecurityInfo property**

Gets or sets the PersistSecurityInfo connection property.

Syntax

**Visual Basic**

Public Property **PersistSecurityInfo** As Boolean

**C#**

public bool **PersistSecurityInfo** { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**Pooling property**

Gets or sets the Pooling connection property.

Syntax

**Visual Basic**

Public Property **Pooling** As Boolean

**C#**

public bool **Pooling** { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269
**PrefetchBuffer property**

Gets or sets the PrefetchBuffer connection property.

**Syntax**

**Visual Basic**

Public Property PrefetchBuffer As Integer

**C#**

public int PrefetchBuffer { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**PrefetchRows property**

Gets or sets the PrefetchRows connection property. The default value is 200.

**Syntax**

**Visual Basic**

Public Property PrefetchRows As Integer

**C#**

public int PrefetchRows { get; set; }

**See also**

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**RetryConnectionTimeout property**

Gets or sets the RetryConnectionTimeout property.

**Syntax**

**Visual Basic**

Public Property RetryConnectionTimeout As Integer

**C#**

public int RetryConnectionTimeout { get; set; }
See also
- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**ServerName property**

Gets or sets the ServerName connection property.

**Syntax**

**Visual Basic**

Public Property ServerName As String

**C#**

public string ServerName { get; set; }

See also
- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**StartLine property**

Gets or sets the StartLine connection property.

**Syntax**

**Visual Basic**

Public Property StartLine As String

**C#**

public string StartLine { get; set; }

See also
- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**Unconditional property**

Gets or sets the Unconditional connection property.
Syntax

**Visual Basic**

Public Property **Unconditional** As String

**C#**

public string **Unconditional** { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**UserID property**

Gets or sets the UserID connection property.

Syntax

**Visual Basic**

Public Property **UserID** As String

**C#**

public string **UserID** { get; set; }

See also

- “SAConnectionStringBuilder class” on page 269
- “SAConnectionStringBuilder members” on page 269

**SAConnectionStringBuilderBase class**

Base class of the SAConnectionStringBuilder class. This class is abstract and so cannot be instantiated.

Syntax

**Visual Basic**

MustInherit Public Class **SAConnectionStringBuilderBase**
Inherits DbConnectionStringBuilder

**C#**

public abstract class **SAConnectionStringBuilderBase** : DbConnectionStringBuilder

See also

- “SAConnectionStringBuilderBase members” on page 292
# SAConnectionStringBuilderBase members

## Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrowsableConnectionString (inherited from DbConnectionStringStringBuilder)</td>
<td>Gets or sets a value that indicates whether the DbConnectionStringBuilder.ConnectionString is visible in Visual Studio designers.</td>
</tr>
<tr>
<td>ConnectionString (inherited from DbConnectionStringBuilder)</td>
<td>Gets or sets the connection string associated with the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td>Count (inherited from DbConnectionStringBuilder)</td>
<td>Gets the current number of keys that are contained within the DbConnectionStringBuilder.ConnectionString.</td>
</tr>
<tr>
<td>IsFixedSize (inherited from DbConnectionStringBuilder)</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder has a fixed size.</td>
</tr>
<tr>
<td>IsReadOnly (inherited from DbConnectionStringBuilder)</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder is read-only.</td>
</tr>
<tr>
<td><strong>Item property</strong></td>
<td>Gets or sets the value of the connection keyword.</td>
</tr>
<tr>
<td><strong>Keys property</strong></td>
<td>Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.</td>
</tr>
<tr>
<td><strong>Values</strong> (inherited from DbConnectionStringBuilder)</td>
<td>Gets an ICollection that contains the values in the DbConnectionStringBuilder.</td>
</tr>
</tbody>
</table>

## Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (inherited from DbConnectionStringBuilder)</td>
<td>Adds an entry with the specified key and value into the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td>Clear (inherited from DbConnectionStringBuilder)</td>
<td>Clears the contents of the DbConnectionStringBuilder instance.</td>
</tr>
<tr>
<td>ContainsKey method</td>
<td>Determines whether the SAConnectionStringBuilder object contains a specific keyword.</td>
</tr>
<tr>
<td>EquivalentTo (inherited from DbConnectionStringBuilder)</td>
<td>Compares the connection information in this DbConnectionStringBuilder object with the connection information in the supplied object.</td>
</tr>
<tr>
<td>GetKeyword method</td>
<td>Gets the keyword for specified SAConnectionStringBuilder property.</td>
</tr>
</tbody>
</table>
## Member name | Description
---|---
GetUseLongNameAsKeyword method | Gets a boolean values that indicates whether long connection parameter names are used in the connection string.
Remove method | Removes the entry with the specified key from the SAConnectionStringBuilder instance.
SetUseLongNameAsKeyword method | Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.
ShouldSerialize method | Indicates whether the specified key exists in this SAConnectionStringBuilder instance.
ToString (inherited from DbConnectionStringBuilder) | Returns the connection string associated with this DbConnectionStringBuilder.
TryGetValue method | Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

### See also
- “SAConnectionStringBuilderBase class” on page 291

## Item property

Gets or sets the value of the connection keyword.

### Syntax

**Visual Basic**

Public Overrides Default Property Item ( _
    ByVal keyword As String _
) As Object

**C#**

public override object this [ string _
    keyword _
] { get; set; }

### Parameters
- **keyword**
The name of the connection keyword.

### Property value

An object representing the value of the specified connection keyword.
Remarks
If the keyword or type is invalid, an exception is raised. Keyword is case insensitive.
When setting the value, passing NULL clears the value.

See also
● “SACConnectionStringBuilderBase class” on page 291
● “SACConnectionStringBuilderBase members” on page 292

Keys property

Gets an System.Collections.ICollection that contains the keys in the SACConnectionStringBuilder.

Syntax
Visual Basic
Public Overrides Readonly Property Keys As ICollection

C#
public override ICollection Keys { get; }

Property value
An System.Collections.ICollection that contains the keys in the SACConnectionStringBuilder.

See also
● “SACConnectionStringBuilderBase class” on page 291
● “SACConnectionStringBuilderBase members” on page 292

ContainsKey method

Determines whether the SACConnectionStringBuilder object contains a specific keyword.

Syntax
Visual Basic
Public Overrides Function ContainsKey( _
    ByVal keyword As String _) As Boolean

C#
public override bool ContainsKey(
    string keyword
);
Parameters

- **keyword**  The keyword to locate in the SAConnectionStringBuilder.

Return value

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.

```csharp
connectString.ContainsKey("UserID")
```

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292

GetKeyword method

Gets the keyword for specified SAConnectionStringBuilder property.

Syntax

**Visual Basic**

Public Function GetKeyword( _
   ByVal propName As String _
) As String

**C#**

public string GetKeyword(
   string propName
);  

Parameters

- **propName**  The name of the SAConnectionStringBuilder property.

Return value

The keyword for specified SAConnectionStringBuilder property.

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292
**GetUseLongNameAsKeyword method**

Gets a boolean values that indicates whether long connection parameter names are used in the connection string.

**Syntax**

**Visual Basic**

Public Function GetUseLongNameAsKeyword() As Boolean

**C#**

public bool GetUseLongNameAsKeyword();

**Return value**

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 class” on page 291
- “SAConnectionStringBuilderBase members” on page 292
- “SetUseLongNameAsKeyword method” on page 297

**Remove method**

Removes the entry with the specified key from the SAConnectionStringBuilder instance.

**Syntax**

**Visual Basic**

Public Overrides Function Remove( _
    ByVal keyword As String _
    ) As Boolean

**C#**

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.
Return value

True if the key existed within the connection string and was removed; false if the key did not exist.

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292

SetUseLongNameAsKeyword method

Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.

Syntax

Visual Basic

Public Sub SetUseLongNameAsKeyword( _
    ByVal useLongNameAsKeyword As Boolean _
) _

C#

public void SetUseLongNameAsKeyword( _
    bool useLongNameAsKeyword _
);

Parameters

- **useLongNameAsKeyword**  A boolean value that indicates whether the long connection parameter name is used in the connection string.

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292
- “GetUseLongNameAsKeyword method” on page 296

ShouldSerialize method

Indicates whether the specified key exists in this SAConnectionStringBuilder instance.

Syntax

Visual Basic

Public Overrides Function ShouldSerialize( _
    ByVal keyword As String _
) As Boolean

C#

iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
public override bool ShouldSerialize(
    string keyword
);

Parameters

- **keyword**  The key to locate in the SAConnectionStringBuilder.

Return value

True if the SAConnectionStringBuilder contains an entry with the specified key; otherwise false.

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292

TryGetValue method

Retrieves a value corresponding to the supplied key from this SAConnectionStringBuilder.

Syntax

**Visual Basic**

Public Overrides Function TryGetValue( _
    ByVal keyword As String, _
    ByVal value As Object _
) As Boolean

**C#**

public override bool TryGetValue(
    string keyword,
    object value
);

Parameters

- **keyword**  The key of the item to retrieve.
- **value**  The value corresponding to keyword.

Return value

true if keyword was found within the connection string; otherwise false.

See also

- “SAConnectionStringBuilderBase class” on page 291
- “SAConnectionStringBuilderBase members” on page 292
**SADDataAdapter class**

Represents a set of commands and a database connection used to fill a `DataSet` and to update a database. This class cannot be inherited.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SADData Adapter
    Inherits DbDataAdapter
```

**C#**

```csharp
public sealed class SADDataAdapter : DbDataAdapter
```

**Remarks**

The `DataSet` provides a way to work with data offline. The SADDataAdapter provides methods to associate a `DataSet` with a set of SQL statements.

**Implements:** `IDbDataAdapter, IDataAdapter, ICloneable`

For more information, see “Using the SADDataAdapter object to access and manipulate data” on page 123 and “Accessing and manipulating data” on page 117.

**See also**

- “SADDataAdapter members” on page 299

**SADDataAdapter members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SADDataAdapter constru</td>
<td>Initializes a new instance of the “SADDataAdapter class” on page 299.</td>
</tr>
</tbody>
</table>

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AcceptChangesDuringFill</code> (inherited from DataAdapter)</td>
<td>Gets or sets a value indicating whether <code>DataRow.AcceptChanges</code> is called on a <code>DataRow</code> after it is added to the <code>DataTable</code> during any of the Fill operations.</td>
</tr>
<tr>
<td><code>AcceptChangesDuringUpdate</code> (inherited from DataAdapter)</td>
<td>Gets or sets whether <code>DataRow.AcceptChanges</code> is called during a <code>DataAdapter.Update</code>.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ContinueUpdateOnError</strong> (inherited from DataAdapter)</td>
<td>Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.</td>
</tr>
<tr>
<td><strong>DeleteCommand property</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>FillLoadOption</strong> (inherited from DataAdapter)</td>
<td>Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.</td>
</tr>
<tr>
<td><strong>InsertCommand property</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>MissingMappingAction</strong> (inherited from DataAdapter)</td>
<td>Determines the action to take when incoming data does not have a matching table or column.</td>
</tr>
<tr>
<td><strong>MissingSchemaAction</strong> (inherited from DataAdapter)</td>
<td>Determines the action to take when existing DataSet schema does not match incoming data.</td>
</tr>
<tr>
<td><strong>ReturnProviderSpecificTypes</strong> (inherited from DataAdapter)</td>
<td>Gets or sets whether the Fill method should return provider-specific values or common CLS-compliant values.</td>
</tr>
<tr>
<td><strong>SelectCommand property</strong></td>
<td>Specifies an SACommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.</td>
</tr>
<tr>
<td><strong>TableMappings property</strong></td>
<td>Specifies a collection that provides the master mapping between a source table and a DataTable.</td>
</tr>
<tr>
<td><strong>UpdateBatchSize property</strong></td>
<td>Gets or sets the number of rows that are processed in each round-trip to the server.</td>
</tr>
<tr>
<td><strong>UpdateCommand property</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fill</strong> (inherited from DbDataAdapter)</td>
<td>Adds or refreshes rows in the DataSet.</td>
</tr>
<tr>
<td><strong>FillSchema</strong> (inherited from DbDataAdapter)</td>
<td>Adds a DataTable named &quot;Table&quot; to the specified DataSet and configures the schema to match that in the data source based on the specified SchemaType.</td>
</tr>
</tbody>
</table>
### Member name

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFillParameters method</td>
<td>Returns the parameters set by you when executing a SELECT statement.</td>
</tr>
<tr>
<td>ResetFillLoadOption (inherited from DataAdapter)</td>
<td>Resets DataAdapter.FillLoadOption to its default state and causes DataAdapter.Fill to honor DataAdapter.AcceptChangesDuringFill.</td>
</tr>
<tr>
<td>ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)</td>
<td>Determines whether the DataAdapter.AcceptChangesDuringFill should be persisted.</td>
</tr>
<tr>
<td>ShouldSerializeFillLoadOption (inherited from DataAdapter)</td>
<td>Determines whether the DataAdapter.FillLoadOption should be persisted.</td>
</tr>
<tr>
<td>Update (inherited from DataAdapter)</td>
<td>Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of DataRow objects.</td>
</tr>
</tbody>
</table>

### Public events

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillError (inherited from DataAdapter)</td>
<td>Returned when an error occurs during a fill operation.</td>
</tr>
<tr>
<td>RowUpdated event</td>
<td>Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.</td>
</tr>
<tr>
<td>RowUpdating event</td>
<td>Occurs during an update before a command is executed against the data source. When an attempt to update is made, the event fires.</td>
</tr>
</tbody>
</table>

### See also

- “SADatabase class” on page 299

### SADatabase constructors

Initializes a new instance of the “SADatabase class” on page 299.

### SADatabase() constructor

Initializes an SADatabase object.
Synta

Visual Basic
Public Sub New()

C#
public SDataAdapter();

See also
- “SDataAdapter class” on page 299
- “SDataAdapter members” on page 299
- “SDataAdapter constructors” on page 301
- “SDataAdapter(SACommand) constructor” on page 302
- “SDataAdapter(String, SAConnection) constructor” on page 303
- “SDataAdapter(String, String) constructor” on page 303

SDataAdapter(SACommand) constructor

Initializes an SDataAdapter object with the specified SELECT statement.

Syntax

Visual Basic
Public Sub New(
    ByVal selectCommand As SACommand
)

C#
public SDataAdapter(
    SACommand selectCommand
);
**SADDataAdapter(String, SAConnection) constructor**

Initializes an SADDataAdapter object with the specified SELECT statement and connection.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New( _
    ByVal selectCommandText As String, _
    ByVal selectConnection As SAConnection _
)
```

**C#**

```csharp
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 class” on page 299
- “SADDataAdapter members” on page 299
- “SADDataAdapter constructors” on page 301
- “SDataAdapter() constructor” on page 301
- “SADDataAdapter(SACommand) constructor” on page 302
- “SADDataAdapter(String, String) constructor” on page 303
- “SelectCommand property” on page 305
- “SAConnection class” on page 246

**SADDataAdapter(String, String) constructor**

Initializes an SADDataAdapter object with the specified SELECT statement and connection string.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New( _
    ByVal selectCommandText As String, _
    ByVal selectConnectionString As String _
)
```

**C#**

```csharp
public SADDataAdapter(  
    string selectCommandText,  
    string selectConnectionString  
);
string selectCommandText, 
string selectConnectionString 
);

Parameters

- **selectCommandText**  A SELECT statement to be used to set the SADeleteAdapter.SelectCommand property of the SADeleteAdapter object.
- **selectConnectionString**  A connection string for a SQL Anywhere database.

See also

- “SADeleteAdapter class” on page 299
- “SADeleteAdapter members” on page 299
- “SADeleteAdapter constructors” on page 301
- “SADeleteAdapter() constructor” on page 301
- “SADeleteAdapter(SACommand) constructor” on page 302
- “SADeleteAdapter(String, SAConnection) constructor” on page 303
- “SelectCommand property” on page 305

**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.

**Syntax**

**Visual Basic**

Public Property DeleteCommand As SACommand

**C#**

public 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

- “SADeleteAdapter class” on page 299
- “SADeleteAdapter members” on page 299
- “SelectCommand property” on page 305
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.

Syntax

Visual Basic

Public Property InsertCommand As SACommand

C#

public 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.

See also

- “SADataAdapter class” on page 299
- “SADataAdapter members” on page 299

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.

Syntax

Visual Basic

Public Property SelectCommand As SACommand

C#

public 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 SADatasAdapter constructor.
See also
- “SADataAdapter class” on page 299
- “SADataAdapter members” on page 299

**TableMappings property**

Specifies a collection that provides the master mapping between a source table and a DataTable.

**Syntax**

**Visual Basic**

```vbnet
Public Readonly Property TableMappings As DataTableMappingCollection
```

**C#**

```csharp
public DataTableMappingCollection TableMappings { get; }
```

**Remarks**

The default value is an empty collection.

When reconciling changes, the SADataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

**Restrictions:** The TableMappings property is not available in the .NET Compact Framework 2.0.

See also
- “SADataAdapter class” on page 299
- “SADataAdapter members” on page 299

**UpdateBatchSize property**

Gets or sets the number of rows that are processed in each round-trip to the server.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Property UpdateBatchSize As Integer
```

**C#**

```csharp
public override int UpdateBatchSize { get; set; }
```

**Remarks**

The default value is 1.

Setting the value to something greater than 1 causes SADataAdapter.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.

See also

- “SADDataAdapter class” on page 299
- “SADDataAdapter members” on page 299

### UpdateCommand property

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.

**Syntax**

**Visual Basic**

Public Property UpdateCommand As SACommand

**C#**

public SACommand 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.

See also

- “SADDataAdapter class” on page 299
- “SADDataAdapter members” on page 299
getFillParameters method

Returns the parameters set by you when executing a SELECT statement.

Syntax

Visual Basic

Public Function getFillParameters() As SAParameter

C#

public SAParameter getFillParameters();

Return value

An array of IDataParameter objects that contains the parameters set by the user.

See also

● “SADataset class” on page 299
● “SADataset members” on page 299

RowUpdated event

Occurs during an update after a command is executed against the data source. When an attempt to update is made, the event fires.

Syntax

Visual Basic

Public Event RowUpdated As SARowUpdatedEventHandler

C#

public event SARowUpdatedEventHandler RowUpdated ;

Remarks

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.

Event data

● Command Gets the SACommand that is executed whenDataAdapter.Update is called.

● RecordsAffected Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

● Command Gets the IDbCommand executed when DbDataAdapter.Update is called.
● **Errors**  Gets any errors generated by the .NET Framework data provider when the
  `RowUpdatedEventArgs.Command` was executed.

● **Row**  Gets the `DataRow` sent through an `DbDataAdapter.Update`.

● **RowCount**  Gets the number of rows processed in a batch of updated records.

● **StatementType**  Gets the type of SQL statement executed.

● **Status**  Gets the `UpdateStatus` of the `RowUpdatedEventArgs.Command`.

● **TableMapping**  Gets the `DataTableMapping` sent through an `DbDataAdapter.Update`.

**See also**

- “SADDataAdapter class” on page 299
- “SADDataAdapter members” on page 299

**RowUpdating event**

Occurs during an update before a command is executed against the data source. When an attempt to update
is made, the event fires.

**Syntax**

**Visual Basic**

Public Event RowUpdating As SARowUpdatingEventHandler

**C#**

public event SARowUpdatingEventHandler RowUpdating ;

**Remarks**

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.

**Event data**

- **Command**  Specifies the SACommand to execute when performing the Update.

- **Command**  Gets the IDbCommand to execute during the `DbDataAdapter.Update` operation.

- **Errors**  Gets any errors generated by the .NET Framework data provider when the
  `RowUpdatedEventArgs.Command` executes.

- **Row**  Gets the `DataRow` that will be sent to the server as part of an insert, update, or delete operation.

- **StatementType**  Gets the type of SQL statement to execute.

- **Status**  Gets or sets the `UpdateStatus` of the `RowUpdatedEventArgs.Command`.

- **TableMapping**  Gets the `DataTableMapping` to send through the `DbDataAdapter.Update`.
See also

- “SDataAdapter class” on page 299
- “SDataAdapter members” on page 299

**SDataReader class**

A read-only, forward-only result set from a query or stored procedure. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SDataReader
    Inherits DbDataReader
    Implements IListSource

**C#**

public sealed class SDataReader : DbDataReader,
    IListSource

**Remarks**

There is no constructor for SDataReader. To get an SDataReader object, execute an SACommand:

```csharp
SACommand cmd = new SACommand("SELECT EmployeeID FROM Employees", conn);
SDataReader reader = cmd.ExecuteReader();
```

You can only move forward through an SDataReader. If you need a more flexible object to manipulate results, use an SDataAdapter.

The SDataReader retrieves rows as needed, whereas the SDataAdapter 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 SDataReader a much faster response time.

**Implements:** IDataReader, IDisposable, IDataRecord, IListSource

For more information, see “Accessing and manipulating data” on page 117.

See also

- “SDataReader members” on page 311
- “ExecuteReader() method” on page 226
# SADataReader members

## Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth property</td>
<td>Gets a value indicating the depth of nesting for the current row. The outermost table has a depth of zero.</td>
</tr>
<tr>
<td>FieldCount property</td>
<td>Gets the number of columns in the result set.</td>
</tr>
<tr>
<td>HasRows property</td>
<td>Gets a value that indicates whether the SADataReader contains one or more rows.</td>
</tr>
<tr>
<td>IsClosed property</td>
<td>Gets a value that indicates whether the SADataReader is closed.</td>
</tr>
<tr>
<td>Item properties</td>
<td>Returns the value of a column in its native format. In C#, this property is the indexer for the SADataReader class.</td>
</tr>
<tr>
<td>RecordsAffected property</td>
<td>The number of rows changed, inserted, or deleted by execution of the SQL statement.</td>
</tr>
<tr>
<td>VisibleFieldCount (inherited from DbDataReader)</td>
<td>Gets the number of fields in the DbDataReader that are not hidden.</td>
</tr>
</tbody>
</table>

## Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Closes the SADataReader.</td>
</tr>
<tr>
<td>Dispose (inherited from DbDataReader)</td>
<td>Releases all resources used by the current instance of the DbDataReader.</td>
</tr>
<tr>
<td>GetBoolean method</td>
<td>Returns the value of the specified column as a Boolean.</td>
</tr>
<tr>
<td>GetByte method</td>
<td>Returns the value of the specified column as a Byte.</td>
</tr>
<tr>
<td>GetBytes method</td>
<td>Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.</td>
</tr>
<tr>
<td>GetChar method</td>
<td>Returns the value of the specified column as a character.</td>
</tr>
<tr>
<td>GetChars method</td>
<td>Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.</td>
</tr>
<tr>
<td>GetData method</td>
<td>This method is not supported. When called, it throws an InvalidOperationException.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetDataTypeName method</td>
<td>Returns the name of the source data type.</td>
</tr>
<tr>
<td>GetDateTime method</td>
<td>Returns the value of the specified column as a DateTime object.</td>
</tr>
<tr>
<td>GetDecimal method</td>
<td>Returns the value of the specified column as a Decimal object.</td>
</tr>
<tr>
<td>GetDouble method</td>
<td>Returns the value of the specified column as a double-precision floating point number.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns a IEnumerator that iterates through the SADataReader object.</td>
</tr>
<tr>
<td>GetFieldType method</td>
<td>Returns the Type that is the data type of the object.</td>
</tr>
<tr>
<td>GetFloat method</td>
<td>Returns the value of the specified column as a single-precision floating point number.</td>
</tr>
<tr>
<td>GetGuid method</td>
<td>Returns the value of the specified column as a global unique identifier (GUID).</td>
</tr>
<tr>
<td>GetInt16 method</td>
<td>Returns the value of the specified column as a 16-bit signed integer.</td>
</tr>
<tr>
<td>GetInt32 method</td>
<td>Returns the value of the specified column as a 32-bit signed integer.</td>
</tr>
<tr>
<td>GetInt64 method</td>
<td>Returns the value of the specified column as a 64-bit signed integer.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the specified column.</td>
</tr>
<tr>
<td>GetOrdinal method</td>
<td>Returns the column ordinal, given the column name.</td>
</tr>
<tr>
<td>GetProviderSpecificFieldType (inherited from DbDataReader)</td>
<td>Returns the provider-specific field type of the specified column.</td>
</tr>
<tr>
<td>GetProviderSpecificValue (inherited from DbDataReader)</td>
<td>Gets the value of the specified column as an instance of Object.</td>
</tr>
<tr>
<td>GetProviderSpecificValues (inherited from DbDataReader)</td>
<td>Gets all provider-specific attribute columns in the collection for the current row.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Returns a DataTable that describes the column metadata of the SADataReader.</td>
</tr>
<tr>
<td>GetString method</td>
<td>Returns the value of the specified column as a string.</td>
</tr>
<tr>
<td>GetTimeSpan method</td>
<td>Returns the value of the specified column as a TimeSpan object.</td>
</tr>
<tr>
<td>GetUInt16 method</td>
<td>Returns the value of the specified column as a 16-bit unsigned integer.</td>
</tr>
</tbody>
</table>
### Depth property

Gets a value indicating the depth of nesting for the current row. The outermost table has a depth of zero.

### Syntax

**Visual Basic**

```vbnet
Public Overrides Readonly Property Depth As Integer
```

**C#**

```csharp
public override int Depth { get; }
```

### Property value

The depth of nesting for the current row.

### See also

- “SADataReader class” on page 310
- “ExecuteReader() method” on page 226

### Table: Member Descriptions

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUInt32 method</td>
<td>Returns the value of the specified column as a 32-bit unsigned integer.</td>
</tr>
<tr>
<td>GetUInt64 method</td>
<td>Returns the value of the specified column as a 64-bit unsigned integer.</td>
</tr>
<tr>
<td>GetValue methods</td>
<td>Returns the value of the specified column as an Object.</td>
</tr>
<tr>
<td>GetValues method</td>
<td>Gets all the columns in the current row.</td>
</tr>
<tr>
<td>IsDBNull method</td>
<td>Returns a value indicating whether the column contains NULL values.</td>
</tr>
<tr>
<td>NextResult method</td>
<td>Advances the SADataReader to the next result, when reading the results of batch SQL statements.</td>
</tr>
<tr>
<td>Read method</td>
<td>Reads the next row of the result set and moves the SADataReader to that row.</td>
</tr>
<tr>
<td>myDispose method</td>
<td>Frees the resources associated with the object.</td>
</tr>
</tbody>
</table>

See also

- “SADataReader class” on page 310
- “ExecuteReader() method” on page 226
FieldCount property

Gets the number of columns in the result set.

Syntax

Visual Basic
Public Overrides Readonly Property FieldCount As Integer

C#
public override int FieldCount { get; }

Property value

The number of columns in the current record.

See also

● “SADataReader class” on page 310
● “SADataReader members” on page 311

HasRows property

Gets a value that indicates whether the SADataReader contains one or more rows.

Syntax

Visual Basic
Public Overrides Readonly Property HasRows As Boolean

C#
public override bool HasRows { get; }

Property value

True if the SADataReader contains one or more rows; otherwise, false.

See also

● “SADataReader class” on page 310
● “SADataReader members” on page 311

IsClosed property

Gets a values that indicates whether the SADataReader is closed.
Syntax

Visual Basic

Public Overrides Readonly Property IsClosed As Boolean

C#

public override bool IsClosed { get; }

Property value

True if the SADataReader is closed; otherwise, false.

Remarks

IsClosed and RecordsAffected are the only properties that you can call after the SADataReader is closed.

See also

● “SADataReader class” on page 310
● “SADataReader members” on page 311

Item properties

Returns the value of a column in its native format. In C#, this property is the indexer for the SADataReader class.

Item(Int32) property

Returns the value of a column in its native format. In C#, this property is the indexer for the SADataReader class.

Syntax

Visual Basic

Public Overrides Default Readonly Property Item ( _
     ByVal index As Integer _
) As Object

C#

public override object this [ int index
    ] { get; }
Item(String) property

Returns the value of a column in its native format. In C#, this property is the indexer for the SADeReader class.

Syntax

**Visual Basic**

```
Public Overrides Default Readonly Property Item ( _
  ByVal name As String _
) As Object
```

**C#**

```
public override object this [ 
  string name
] { get;}
```

Parameters

- **name**  The column name.

See also

- “SADeReader class” on page 310
- “SADeReader members” on page 311
- “Item properties” on page 315

RecordsAffected property

The number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax

**Visual Basic**

```
Public Overrides Readonly Property RecordsAffected As Integer
```

**C#**

```
public override int RecordsAffected { get;}
```

Property value

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.
Remarks
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 SADataReader is closed.

See also
- “SADataReader class” on page 310
- “SADataReader members” on page 311

Close method
Closes the SADataReader.

Syntax
Visual Basic
Public Overrides Sub Close()

C#
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.

See also
- “SADataReader class” on page 310
- “SADataReader members” on page 311

GetBoolean method
Returns the value of the specified column as a Boolean.

Syntax
Visual Basic
Public Overrides Function GetBoolean( _
    ByVal ordinal As Integer _
) As Boolean

C#
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.

Return value

The value of the column.

Remarks

No conversions are performed, so the data retrieved must already be a Boolean.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311
- “GetOrdinal method” on page 330
- “GetFieldType method” on page 325

GetByte method

Returns the value of the specified column as a Byte.

Syntax

**Visual Basic**

```vbnet
Public Overrides Function GetByte(ByVal ordinal As Integer) As Byte
```

**C#**

```csharp
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.

Return value

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.

Syntax

Visual Basic

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#

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.

Return value

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.

See also
● “SADataReader class” on page 310
● “SADataReader members” on page 311

GetChar method

Returns the value of the specified column as a character.

Syntax

Visual Basic

Public Overrides Function GetChar( _
 ByVal ordinal As Integer _
) As Char

C#

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.

Return value

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 class” on page 310
● “SADataReader members” on page 311
● “IsDBNull method” on page 339
GetChars method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax

Visual Basic

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#

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.

Return value

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 SDataReader 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 “Handling BLOBs” on page 132.
See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311

**GetData method**

This method is not supported. When called, it throws an InvalidOperationException.

**Syntax**

**Visual Basic**

```vbnet
Public Function GetData( _
    ByVal i As Integer _
) As IDataReader
```

**C#**

```csharp
public IDataReader GetData( int i );
```

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311
- InvalidOperationException

**GetDataTypeName method**

Returns the name of the source data type.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Function GetDataTypeName( _
    ByVal index As Integer _
) As String
```

**C#**

```csharp
public override string GetDataTypeName( int index );
```

**Parameters**

- **index** The zero-based column ordinal.
Return value

The name of the back-end data type.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311

GetDateTime method

Returns the value of the specified column as a DateTime object.

Syntax

Visual Basic

Public Overrides Function GetDateTime( _
    ByVal ordinal As Integer _
) As Date

C#

public override DateTime GetDateTime(  
    int ordinal  
);  

Parameters

- ordinal The zero-based column ordinal.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a DateTime object.

Call the SADeataReader.IsDBNull method to check for null values before calling this method.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311
- “IsDBNull method” on page 339

GetDecimal method

Returns the value of the specified column as a Decimal object.
Syntax

Visual Basic

Public Overrides Function GetDecimal( _
    ByVal ordinal As Integer _
) As Decimal

C#

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.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a Decimal object.

Call the SDataReader.IsDBNull method to check for null values before calling this method.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311
- “IsDBNull method” on page 339

GetDouble method

Returns the value of the specified column as a double-precision floating point number.

Syntax

Visual Basic

Public Overrides Function GetDouble( _
    ByVal ordinal As Integer _
) As Double

C#

public override double GetDouble(
    int ordinal
);

SQL Anywhere .NET 2.0 API reference

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

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 SReader.IsDBNull method to check for null values before calling this method.

See also

- “SReader class” on page 310
- “SReader members” on page 311
- “DBNull method” on page 339

GetEnumerator method

Returns a IEnumerator that iterates through the SReader object.

Syntax

**Visual Basic**

Public Overrides Function GetEnumerator() As IEnumerator

**C#**

public override IEnumerator GetEnumerator();

Return value

A IEnumerator for the SReader object.

See also

- “SReader class” on page 310
- “SReader members” on page 311
- “DBNull method” on page 339

GetFieldType method

Returns the Type that is the data type of the object.
Syntax

Visual Basic

Public Overrides Function GetFieldType( ByVal index As Integer ) As Type

C#

public override Type GetFieldType( int index );

Parameters

● index The zero-based column ordinal.

Return value

The type that is the data type of the object.

See also

● “SADataReader class” on page 310
● “SADataReader members” on page 311

GetFloat method

Returns the value of the specified column as a single-precision floating point number.

Syntax

Visual Basic

Public Overrides Function GetFloat( ByVal ordinal As Integer ) As Single

C#

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.

Return value

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 SADeReader.IsDBNull method to check for null values before calling this method.

See also
● “SADeReader class” on page 310
● “SADeReader members” on page 311
● “DBNull method” on page 339

GetGuid method

Returns the value of the specified column as a global unique identifier (GUID).

Syntax
Visual Basic
Public Overrides Function GetGuid( _
   ByVal ordinal As Integer _
) As Guid

C#

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.

Return value
The value of the specified column.

Remarks
The data retrieved must already be a globally-unique identifier or binary(16).
Call the SADeReader.IsDBNull method to check for null values before calling this method.

See also
● “SADeReader class” on page 310
● “SADeReader members” on page 311
● “DBNull method” on page 339
GetInt16 method

Returns the value of the specified column as a 16-bit signed integer.

Syntax

Visual Basic

Public Overrides Function GetInt16( _
    ByVal ordinal As Integer _
) As Short

C#

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.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 16-bit signed integer.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311

GetInt32 method

Returns the value of the specified column as a 32-bit signed integer.

Syntax

Visual Basic

Public Overrides Function GetInt32( _
    ByVal ordinal As Integer _
) As Integer

C#

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.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 32-bit signed integer.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311

GetInt64 method

Returns the value of the specified column as a 64-bit signed integer.

Syntax

**Visual Basic**

```vbnet
Public Overrides Function GetInt64( _
    ByVal ordinal As Integer _
) As Long
```

**C#**

```csharp
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.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 64-bit signed integer.

See also

- “SADataReader class” on page 310
- “SADataReader members” on page 311
**GetName method**

Returns the name of the specified column.

**Syntax**

**Visual Basic**

Public Overrides Function GetName( ByVal index As Integer ) As String

**C#**

public override string GetName( int index );

**Parameters**

- **index** The zero-based index of the column.

**Return value**

The name of the specified column.

**See also**

- “SADataReader class” on page 310
- “SADataReader members” on page 311

---

**GetOrdinal method**

Returns the column ordinal, given the column name.

**Syntax**

**Visual Basic**

Public Overrides Function GetOrdinal( ByVal name As String ) As Integer

**C#**

public override int GetOrdinal( string name );

**Parameters**

- **name** The column name.
Return value
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.

See also
● “SDataReader class” on page 310
● “SDataReader members” on page 311

GetSchemaTable method
Returns a DataTable that describes the column metadata of the SDataReader.

Syntax
Visual Basic
Public Overrides Function GetSchemaTable() As DataTable

C#
public override DataTable GetSchemaTable();

Return value
A DataTable that describes the column metadata.

Remarks
This method returns metadata about each column in the following order:

<table>
<thead>
<tr>
<th>DataTable column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnName</td>
<td>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.</td>
</tr>
<tr>
<td>ColumnOrdinal</td>
<td>The ID of the column. The value is in the range [0, FieldCount -1].</td>
</tr>
<tr>
<td>ColumnSize</td>
<td>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.</td>
</tr>
<tr>
<td><strong>DataTable column</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NumericPrecision</td>
<td>The precision of a numeric column or DBNull if the column is not numeric.</td>
</tr>
<tr>
<td>NumericScale</td>
<td>The scale of a numeric column or DBNull if the column is not numeric.</td>
</tr>
<tr>
<td>IsUnique</td>
<td>True if the column is a non-computed unique column in the table (BaseTableName) it is taken from.</td>
</tr>
<tr>
<td>IsKey</td>
<td>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.</td>
</tr>
<tr>
<td>BaseServerName</td>
<td>The name of the SQL Anywhere database server used by the SADatabaseReader.</td>
</tr>
<tr>
<td>BaseCatalogName</td>
<td>The name of the catalog in the database that contains the column. This value is always DBNull.</td>
</tr>
<tr>
<td>BaseColumnName</td>
<td>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.</td>
</tr>
<tr>
<td>BaseSchemaName</td>
<td>The name of the schema in the database that contains the column.</td>
</tr>
<tr>
<td>BaseTableName</td>
<td>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.</td>
</tr>
<tr>
<td>DataType</td>
<td>The .NET data type that is most appropriate for this type of column.</td>
</tr>
<tr>
<td>AllowDBNull</td>
<td>True if the column is nullable, false if the column is not nullable or if this information cannot be determined.</td>
</tr>
<tr>
<td>ProviderType</td>
<td>The type of the column.</td>
</tr>
<tr>
<td>IsAliased</td>
<td>True if the column name is an alias, false if it is not an alias.</td>
</tr>
<tr>
<td>IsExpression</td>
<td>True if the column is an expression, false if it is a column value.</td>
</tr>
<tr>
<td>IsIdentity</td>
<td>True if the column is an identity column, false if it is not an identity column.</td>
</tr>
<tr>
<td>IsAutoIncrement</td>
<td>True if the column is an autoincrement or global autoincrement column, false otherwise (or if this information cannot be determined).</td>
</tr>
</tbody>
</table>
### DataTable column

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsRowVersion</td>
<td>True if the column contains a persistent row identifier that cannot be written to, and has no meaningful value except to identify the row.</td>
</tr>
<tr>
<td>IsHidden</td>
<td>True if the column is hidden, false otherwise.</td>
</tr>
<tr>
<td>IsLong</td>
<td>True if the column is a long varchar, long nvarchar, or a long binary column, false otherwise.</td>
</tr>
<tr>
<td>IsReadOnly</td>
<td>True if the column is read-only, false if the column is modifiable or if its access cannot be determined.</td>
</tr>
</tbody>
</table>

For more information about these columns, see the .NET Framework documentation for SqlDataReader.GetSchemaTable.

For more information, see “Obtaining DataReader schema information” on page 122.

**See also**
- “SDataReader class” on page 310
- “SDataReader members” on page 311

## GetString method

Returns the value of the specified column as a string.

### Syntax

**Visual Basic**

```vbnet
Public Overrides Function GetString( _
   ByVal ordinal As Integer _
) As String
```

**C#**

```csharp
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.

### Return value

The value of the specified column.

### Remarks

No conversions are performed, so the data retrieved must already be a string.
Call the SDataReader.IsDBNull method to check for NULL values before calling this method.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311
- “IsDBNull method” on page 339

GetTimeSpan method

Returns the value of the specified column as a TimeSpan object.

Syntax

Visual Basic

Public Function GetTimeSpan( _
    ByVal ordinal As Integer _
) As TimeSpan

C#

public TimeSpan GetTimeSpan(
    int ordinal
);

Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

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 SDataReader.IsDBNull method to check for NULL values before calling this method.

For more information, see “Obtaining time values” on page 132.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311
- “IsDBNull method” on page 339
GetUInt16 method

Returns the value of the specified column as a 16-bit unsigned integer.

Syntax

Visual Basic

Public Function GetUInt16( _
    ByVal ordinal As Integer  _
) As UInt16

C#

public ushort GetUInt16(
    int ordinal
);

Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 16-bit unsigned integer.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311

GetUInt32 method

Returns the value of the specified column as a 32-bit unsigned integer.

Syntax

Visual Basic

Public Function GetUInt32( _
    ByVal ordinal As Integer  _
) As UInt32

C#

public uint GetUInt32(
    int ordinal
);
Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 32-bit unsigned integer.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311

GetUInt64 method

Returns the value of the specified column as a 64-bit unsigned integer.

Syntax

Visual Basic

```vbnet
Public Function GetUInt64(  
    ByVal ordinal As Integer  
) As UInt64
```

C#

```csharp
public ulong GetUInt64(  
    int ordinal  
);
```

Parameters

- **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

The value of the specified column.

Remarks

No conversions are performed, so the data retrieved must already be a 64-bit unsigned integer.

See also

- “SDataReader class” on page 310
- “SDataReader members” on page 311
**GetValue methods**

Returns the value of the specified column as an Object.

**GetValue(Int32) method**

Returns the value of the specified column as an Object.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Function GetValue(ByVal ordinal As Integer) As Object
```

**C#**

```csharp
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.

**Return value**

The value of the specified column as an object.

**Remarks**

This method returns DBNull for NULL database columns.

**See also**

- “SDataReader class” on page 310
- “SDataReader members” on page 311
- “GetValue methods” on page 337

**GetValue(Int32, Int64, Int32) method**

Returns a substring of the value of the specified column as an Object.

**Syntax**

**Visual Basic**

```vbnet
Public Function GetValue(ByVal ordinal As Integer, ByVal index As Long) As Object
```

**C#**

```csharp
public object GetValue(int ordinal, long index);
```
ByVal length As Integer
) As Object

C#

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.

Return value

The substring value is returned as an object.

Remarks

This method returns DBNull for NULL database columns.

See also

● “SADataReader class” on page 310
● “SADataReader members” on page 311
● “GetValue methods” on page 337

GetValues method

Gets all the columns in the current row.

Syntax

Visual Basic

Public Overrides Function GetValues( _
    ByVal values As Object() _
) As Integer

C#

public override int GetValues(
    object[] values
);

Parameters

● values An array of objects that holds an entire row of the result set.
Return value

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.

See also

● “SDataReader class” on page 310
● “SDataReader members” on page 311

IsDBNull method

Returns a value indicating whether the column contains NULL values.

Syntax

Visual Basic

Public Overrides Function IsDBNull( _
    ByVal ordinal As Integer _
) As Boolean

C#

public override bool IsDBNull(
    int ordinal
);

Parameters

● ordinal The zero-based column ordinal.

Return value

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.

See also

● “SDataReader class” on page 310
● “SDataReader members” on page 311
**NextResult method**

Advances the SADataReader to the next result, when reading the results of batch SQL statements.

**Syntax**

**Visual Basic**

Public Overrides Function NextResult() As Boolean

**C#**

public override bool NextResult();

**Return value**

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.

**See also**

- “SADataReader class” on page 310
- “SADataReader members” on page 311

**Read method**

Reads the next row of the result set and moves the SADataReader to that row.

**Syntax**

**Visual Basic**

Public Overrides Function Read() As Boolean

**C#**

public override bool Read();

**Return value**

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();

See also
  ● “SDataReader class” on page 310
  ● “SDataReader members” on page 311

**myDispose method**

Frees the resources associated with the object.

**Syntax**

**Visual Basic**

Public Sub myDispose()

**C#**

public void myDispose();

See also
  ● “SDataReader class” on page 310
  ● “SDataReader members” on page 311

**SADataSourceEnumerator class**

Provides a mechanism for enumerating all available instances of SQL Anywhere database servers within the local network. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SADataSourceEnumerator
  Inherits DbDataSourceEnumerator

**C#**

public sealed class SADataSourceEnumerator : DbDataSourceEnumerator

**Remarks**

There is no constructor for SADataSourceEnumerator.

The SADataSourceEnumerator class is not available in the .NET Compact Framework 2.0.
See also

- “SADatasourceEnumerator members” on page 342

**SADatasourceEnumerator members**

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance property</td>
<td>Gets an instance of SADatasourceEnumerator, which can be used to retrieve information about all visible SQL Anywhere database servers.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDataSources method</td>
<td>Retrieves a DataTable containing information about all visible SQL Anywhere database servers.</td>
</tr>
</tbody>
</table>

See also

- “SADatasourceEnumerator class” on page 341

**Instance property**

Gets an instance of SADatasourceEnumerator, which can be used to retrieve information about all visible SQL Anywhere database servers.

**Syntax**

**Visual Basic**

Public Shared Readonly Property Instance As SADatasourceEnumerator

**C#**

public const SADatasourceEnumerator Instance { get; }  

See also

- “SADatasourceEnumerator class” on page 341
- “SADatasourceEnumerator members” on page 342

**GetDataSources method**

Retrieves a DataTable containing information about all visible SQL Anywhere database servers.
Syntax

**Visual Basic**

Public Overrides Function **GetDataSources**() As DataTable

**C#**

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.

```csharp
DataTable servers = SADatabaseEnumerator.Instance.GetDataSources();
```

See also

- “SADatabaseEnumerator class” on page 341
- “SADatabaseEnumerator members” on page 342

**SADbType enumeration**

Enumerates the SQL Anywhere .NET database data types.

Syntax

**Visual Basic**

Public Enum **SADbType**

**C#**

public enum **SADbType**

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.

<table>
<thead>
<tr>
<th>SADbType</th>
<th>Compatible .NET type</th>
<th>C# built-in type</th>
<th>Visual Basic built-in type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>System.Int64</td>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>SADbType</td>
<td>Compatible .NET type</td>
<td>C# built-in type</td>
<td>Visual Basic built-in type</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------</td>
<td>-----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Binary, VarBinary</td>
<td>System.Byte[], or System.Guid if size is 16</td>
<td>byte[]</td>
<td>Byte()</td>
</tr>
<tr>
<td>Bit</td>
<td>System.Boolean</td>
<td>bool</td>
<td>Boolean</td>
</tr>
<tr>
<td>Char, VarChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Date</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>Date/Time-Stand</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>Decimal, Numeric</td>
<td>System.String</td>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>Double</td>
<td>System.Double</td>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>Float, Real</td>
<td>System.Single</td>
<td>float</td>
<td>Single</td>
</tr>
<tr>
<td>Image</td>
<td>System.Byte[]</td>
<td>byte[]</td>
<td>Byte()</td>
</tr>
<tr>
<td>Integer</td>
<td>System.Int32</td>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>LongBinary</td>
<td>System.Byte[]</td>
<td>byte[]</td>
<td>Byte()</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Money</td>
<td>System.String</td>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>NChar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>NText</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Numeric</td>
<td>System.String</td>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>NVarchar</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>SmallDateTime</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>SmallInt</td>
<td>System.Int16</td>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>SADbType</td>
<td>Compatible .NET type</td>
<td>C# built-in type</td>
<td>Visual Basic built-in type</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>SmallMoney</td>
<td>System.String</td>
<td>decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>SysName</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Text</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Time</td>
<td>System.TimeSpan</td>
<td>TimeSpan (no built-in type)</td>
<td>TimeSpan (no built-in type)</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>System.DateTime</td>
<td>DateTime (no built-in type)</td>
<td>Date</td>
</tr>
<tr>
<td>TinyInt</td>
<td>System.Byte</td>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>UniqueIdentifier</td>
<td>System.Gid</td>
<td>Guid (no built-in type)</td>
<td>Guid (no built-in type)</td>
</tr>
<tr>
<td>UniqueIdentifierStr</td>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>UnsignedBigInt</td>
<td>System.UInt64</td>
<td>ulong</td>
<td>UInt64 (no built-in type)</td>
</tr>
<tr>
<td>UnsignedInt</td>
<td>System.UInt32</td>
<td>uint</td>
<td>UInt64 (no built-in type)</td>
</tr>
<tr>
<td>UnsignedSmallInt</td>
<td>System.UInt16</td>
<td>ushort</td>
<td>UInt64 (no built-in type)</td>
</tr>
<tr>
<td>Xml</td>
<td>System.System.Xml</td>
<td>String</td>
<td>String</td>
</tr>
</tbody>
</table>

Binary columns of length 16 are fully compatible with the UniqueIdentifier type.

**Members**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>Signed 64-bit integer.</td>
<td>1</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary data, with a specified maximum length. The enumeration values Binary and VarBinary are aliases of each other.</td>
<td>2</td>
</tr>
<tr>
<td>Bit</td>
<td>1-bit flag.</td>
<td>3</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>Char</td>
<td>Character data, with a specified length. This type always supports Unicode characters. The types Char and VarChar are fully compatible.</td>
<td>4</td>
</tr>
<tr>
<td>Date</td>
<td>Date information.</td>
<td>5</td>
</tr>
<tr>
<td>DateTime</td>
<td>Timestamp information (date, time). The enumeration values DateTime and TimeStamp are aliases of each other.</td>
<td>6</td>
</tr>
<tr>
<td>Decimal</td>
<td>Exact numerical data, with a specified precision and scale. The enumeration values Decimal and Numeric are aliases of each other.</td>
<td>7</td>
</tr>
<tr>
<td>Double</td>
<td>Double precision floating-point number (8 bytes).</td>
<td>8</td>
</tr>
<tr>
<td>Float</td>
<td>Single precision floating-point number (4 bytes). The enumeration values Float and Real are aliases of each other.</td>
<td>9</td>
</tr>
<tr>
<td>Image</td>
<td>Stores binary data of arbitrary length.</td>
<td>10</td>
</tr>
<tr>
<td>Integer</td>
<td>Unsigned 32-bit integer.</td>
<td>11</td>
</tr>
<tr>
<td>LongBinary</td>
<td>Binary data, with variable length.</td>
<td>12</td>
</tr>
<tr>
<td>LongNVarchar</td>
<td>Character data in the NCHAR character set, with variable length. This type always supports Unicode characters.</td>
<td>13</td>
</tr>
<tr>
<td>LongVarbit</td>
<td>Bit arrays, with variable length.</td>
<td>14</td>
</tr>
<tr>
<td>LongVarchar</td>
<td>Character data, with variable length. This type always supports Unicode characters.</td>
<td>15</td>
</tr>
<tr>
<td>Money</td>
<td>Monetary data.</td>
<td>16</td>
</tr>
<tr>
<td>NChar</td>
<td>Stores Unicode character data, up to 8191 characters.</td>
<td>17</td>
</tr>
<tr>
<td>NText</td>
<td>Stores Unicode character data of arbitrary length.</td>
<td>18</td>
</tr>
<tr>
<td>Numeric</td>
<td>Exact numerical data, with a specified precision and scale. The enumeration values Decimal and Numeric are aliases of each other.</td>
<td>19</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NVarChar</td>
<td>Stores Unicode character data, up to 8191 characters.</td>
<td>20</td>
</tr>
<tr>
<td>Real</td>
<td>Single precision floating-point number (4 bytes). The enumeration values Float and Real are aliases of each other.</td>
<td>21</td>
</tr>
<tr>
<td>SmallDateTime</td>
<td>A domain, implemented as TIMESTAMP.</td>
<td>22</td>
</tr>
<tr>
<td>SmallInt</td>
<td>Signed 16-bit integer.</td>
<td>23</td>
</tr>
<tr>
<td>SmallMoney</td>
<td>Stores monetary data that is less than one million currency units.</td>
<td>24</td>
</tr>
<tr>
<td>SysName</td>
<td>Stores character data of arbitrary length.</td>
<td>25</td>
</tr>
<tr>
<td>Text</td>
<td>Stores character data of arbitrary length.</td>
<td>26</td>
</tr>
<tr>
<td>Time</td>
<td>Time information.</td>
<td>27</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Timestamp information (date, time). The enumeration values DateTime and TimeStamp are aliases of each other.</td>
<td>28</td>
</tr>
<tr>
<td>TinyInt</td>
<td>Unsigned 8-bit integer.</td>
<td>29</td>
</tr>
<tr>
<td>UniqueIdentifier</td>
<td>Universally Unique Identifier (UUID/GUID).</td>
<td>30</td>
</tr>
<tr>
<td>UniqueIdentifierStr</td>
<td>A domain, implemented as CHAR( 36 ). UniqueIdentifierStr is used for remote data access when mapping Microsoft SQL Server uniqueidentifier columns.</td>
<td>31</td>
</tr>
<tr>
<td>UnsignedBigInt</td>
<td>Unsigned 64-bit integer.</td>
<td>32</td>
</tr>
<tr>
<td>UnsignedInt</td>
<td>Unsigned 32-bit integer.</td>
<td>33</td>
</tr>
<tr>
<td>UnsignedSmallInt</td>
<td>Unsigned 16-bit integer.</td>
<td>34</td>
</tr>
<tr>
<td>VarBinary</td>
<td>Binary data, with a specified maximum length. The enumeration values Binary and VarBinary are aliases of each other.</td>
<td>35</td>
</tr>
<tr>
<td>VarBit</td>
<td>Bit arrays that are from 1 to 32767 bits in length.</td>
<td>36</td>
</tr>
</tbody>
</table>
###VarChar
Character data, with a specified maximum length. This type always supports Unicode characters. The types Char and VarChar are fully compatible.

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarChar</td>
<td>Character data, with a specified maximum length. This type always supports Unicode characters. The types Char and VarChar are fully compatible.</td>
<td>37</td>
</tr>
</tbody>
</table>

###Xml
XML data. This type stores character data of arbitrary length, and is used to store XML documents.

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xml</td>
<td>XML data. This type stores character data of arbitrary length, and is used to store XML documents.</td>
<td>38</td>
</tr>
</tbody>
</table>

See also
- “GetFieldType method” on page 325
- “GetDataTypeName method” on page 322

###SADefault class
Represents a parameter with a default value. This is a static class and so cannot be inherited or instantiated.

**Syntax**

**Visual Basic**

```
Public NotInheritable Class SADefault
```

**C#**

```
public sealed class SADefault
```

**Remarks**

There is no constructor for SADefault.

```
SAParameter parm = new SAParameter();
parm.Value = SADefault.Value;
```

See also
- “SADefault members” on page 348

###SADefault members

**Public fields**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value field</td>
<td>Gets the value for a default parameter. This field is read-only and static. This field is read-only.</td>
</tr>
</tbody>
</table>
Value field

Gets the value for a default parameter. This field is read-only and static. This field is read-only.

Syntax

Visual Basic

Public Shared Readonly Value As SADefault

C#

public const SADefault Value ;

See also

● “SADefault class” on page 348
● “SADefault members” on page 348

SAError class

Collects information relevant to a warning or error returned by the data source. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SAError

C#

public sealed class SAError

Remarks

There is no constructor for SAError.

For information about error handling, see “Error handling and the SQL Anywhere .NET Data Provider” on page 138.

See also

● “SAError members” on page 350
# SAEError members

## Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message property</td>
<td>Returns a short description of the error.</td>
</tr>
<tr>
<td>NativeError property</td>
<td>Returns database-specific error information.</td>
</tr>
<tr>
<td>Source property</td>
<td>Returns the name of the provider that generated the error.</td>
</tr>
<tr>
<td>SqlState property</td>
<td>The SQL Anywhere five-character SQLSTATE following the ANSI SQL standard.</td>
</tr>
</tbody>
</table>

## Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToString method</td>
<td>The complete text of the error message.</td>
</tr>
</tbody>
</table>

## See also

- “SAError class” on page 349

## Message property

Returns a short description of the error.

### Syntax

**Visual Basic**

Public Readonly Property `Message` As String

**C#**

```csharp
public string Message { get; }
```

## NativeError property

Returns database-specific error information.

## See also

- “SAError class” on page 349
- “SAError members” on page 350
Syntax
Visual Basic
Public Readonly Property NativeError As Integer

C#
public int NativeError { get; }

See also
● “SAError class” on page 349
● “SAError members” on page 350

Source property

Returns the name of the provider that generated the error.

Syntax
Visual Basic
Public Readonly Property Source As String

C#
public string Source { get; }

See also
● “SAError class” on page 349
● “SAError members” on page 350

SqlState property

The SQL Anywhere five-character SQLSTATE following the ANSI SQL standard.

Syntax
Visual Basic
Public Readonly Property SqlState As String

C#
public string SqlState { get; }

See also
● “SAError class” on page 349
● “SAError members” on page 350
**ToString method**

The complete text of the error message.

**Syntax**

**Visual Basic**

Public Overrides Function **ToString()** As String

**C#**

public override string **ToString();**

**Example**

The return value is a string is in the form **SAError:**, followed by the Message. For example:

SAError:UserId or Password not valid.

**See also**

- “SAError class” on page 349
- “SAError members” on page 350

**SAErrorCollection class**

Collects all errors generated by the SQL Anywhere .NET Data Provider. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class **SAErrorCollection**
Implements ICollection, IEnumerable

**C#**

public sealed class **SAErrorCollection** : ICollection, IEnumerable

**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 138.
See also

- “SAErrorCollection members” on page 353
- “Errors property” on page 357
- SqlClientFactory.CanCreateDataSourceEnumerator

**SAErrorCollection members**

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Returns the number of errors in the collection.</td>
</tr>
<tr>
<td>Item property</td>
<td>Returns the error at the specified index.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyTo method</td>
<td>Copies the elements of the SAErrorCollection into an array, starting at the given index within the array.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an enumerator that iterates through the SAErrorCollection.</td>
</tr>
</tbody>
</table>

See also

- “SAErrorCollection class” on page 352
- “Errors property” on page 357
- SqlClientFactory.CanCreateDataSourceEnumerator

**Count property**

Returns the number of errors in the collection.

**Syntax**

**Visual Basic**

```
NotOverridable Public Readonly Property Count As Integer
```

**C#**

```
public int Count { get; }
```

See also

- “SAErrorCollection class” on page 352
- “SAErrorCollection members” on page 353
**Item property**

Returns the error at the specified index.

**Syntax**

**Visual Basic**

Public Readonly Property Item ( _
    ByVal index As Integer _
) As SAError

**C#**

public SAError this [int index]
    { get; }

**Parameters**

- **index**  The zero-based index of the error to retrieve.

**Property value**

An SAError object that contains the error at the specified index.

**See also**

- “SAErrorCollection class” on page 352
- “SAErrorCollection members” on page 353
- “SAError class” on page 349

**CopyTo method**

Copies the elements of the SAErrorCollection into an array, starting at the given index within the array.

**Syntax**

**Visual Basic**

NotOverridable Public Sub CopyTo( _
    ByVal array As Array, _
    ByVal index As Integer _
)

**C#**

public void CopyTo( 
    Array array,
    int index
);

**Parameters**

- **array**  The array into which to copy the elements.
• **index**  The starting index of the array.

**See also**

- “SAErrorCollection class” on page 352
- “SAErrorCollection members” on page 353

### GetEnumerator method

Returns an enumerator that iterates through the SAErrorCollection.

**Syntax**

**Visual Basic**

```vbnet
NotOverridable Public Function GetEnumerator() As IEnumerator
```

**C#**

```csharp
public IEnumerator GetEnumerator();
```

**Return value**

An `IEnumerator` for the SAErrorCollection.

**See also**

- “SAErrorCollection class” on page 352
- “SAErrorCollection members” on page 353

### SAException class

The exception that is thrown when SQL Anywhere returns a warning or error.

**Syntax**

**Visual Basic**

```vbnet
Public Class SAException
    Inherits DbException
```

**C#**

```csharp
public class SAException : DbException
```

**Remarks**

There is no constructor for SAException. Typically, an SAException object is declared in a catch. For example:

```csharp
... 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 138.

See also
● “SAException members” on page 356

SAException members

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (inherited from Exception)</td>
<td>Gets a collection of key/value pairs that provide additional user-defined information about the exception.</td>
</tr>
<tr>
<td>ErrorCode (inherited from ExternalException)</td>
<td>Gets the HRESULT of the error.</td>
</tr>
<tr>
<td>Errors property</td>
<td>Returns a collection of one or more “SAError class” on page 349 objects.</td>
</tr>
<tr>
<td>HelpLink (inherited from Exception)</td>
<td>Gets or sets a link to the help file associated with this exception.</td>
</tr>
<tr>
<td>InnerException (inherited from Exception)</td>
<td>Gets the Exception instance that caused the current exception.</td>
</tr>
<tr>
<td>Message property</td>
<td>Returns the text describing the error.</td>
</tr>
<tr>
<td>NativeError property</td>
<td>Returns database-specific error information.</td>
</tr>
<tr>
<td>Source property</td>
<td>Returns the name of the provider that generated the error.</td>
</tr>
<tr>
<td>StackTrace (inherited from Exception)</td>
<td>Gets a string representation of the frames on the call stack at the time the current exception was thrown.</td>
</tr>
<tr>
<td>TargetSite (inherited from Exception)</td>
<td>Gets the method that throws the current exception.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBaseException (inherited from Exception)</td>
<td>When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.</td>
</tr>
</tbody>
</table>
### Member name

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetObjectData method</td>
<td>Sets the SerializationInfo with information about the exception. Overrides Exception.GetObjectData.</td>
</tr>
<tr>
<td>GetType (inherited from Exception)</td>
<td>Gets the runtime type of the current instance.</td>
</tr>
<tr>
<td>ToString (inherited from Exception)</td>
<td>Creates and returns a string representation of the current exception.</td>
</tr>
</tbody>
</table>

**See also**

- “SAException class” on page 355

### Errors property

Returns a collection of one or more “SAError class” on page 349 objects.

**Syntax**

**Visual Basic**

Public Readonly Property Errors As SAErrorCollection

**C#**

public SAErrorCollection Errors { get; }

**Remarks**

The SAErrorCollection object always contains at least one instance of the SAError object.

**See also**

- “SAException class” on page 355
- “SAException members” on page 356
- “SAErrorCollection class” on page 352
- “SAError class” on page 349

### Message property

Returns the text describing the error.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property Message As String
C# 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
● “SAException class” on page 355
● “SAException members” on page 356
● “SAError class” on page 349

NativeError property
Returns database-specific error information.

Syntax
Visual Basic
Public Readonly Property NativeError As Integer

C#
public int NativeError { get; }

See also
● “SAException class” on page 355
● “SAException members” on page 356

Source property
Returns the name of the provider that generated the error.

Syntax
Visual Basic
Public Overrides Readonly Property Source As String

C#
public override string Source { get; }
See also
   ● “SAException class” on page 355
   ● “SAException members” on page 356

**GetObjectData method**

Sets the SerializationInfo with information about the exception. Overrides Exception.GetObjectData.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Sub GetObjectData( _
    ByVal info As SerializationInfo, _
    ByVal context As StreamingContext _
)
```

**C#**

```csharp
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.

See also
   ● “SAException class” on page 355
   ● “SAException members” on page 356

**SAFactory class**

Represents a set of methods for creating instances of the iAnywhere.Data.SQLAnywhere provider's implementation of the data source classes. This is a static class and so cannot be inherited or instantiated.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SAFactory
    Inherits DbProviderFactory
    Implements IServiceProvider
```

**C#**

```csharp
public sealed class SAFactory : DbProviderFactory,
    IServiceProvider
```
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:

```vbnet
' 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 SAConnection 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.

Restrictions: The SAFactory class is not available in the .NET Compact Framework 2.0.

Inherits: DbProviderFactory

See also

● “SAFactory members” on page 360

SAFactory members

Public fields

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance field</td>
<td>Represents the singleton instance of the SAFactory class. This field is read-only.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanCreateDataSourceEnumerator property</td>
<td>Always returns true, which indicates that an SADataSourceEnumerator object can be created.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCommand method</td>
<td>Returns a strongly typed DbCommand instance.</td>
</tr>
</tbody>
</table>
### Member name
<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCommandBuilder method</td>
<td>Returns a strongly typed DbCommandBuilder instance.</td>
</tr>
<tr>
<td>CreateConnection method</td>
<td>Returns a strongly typed DbConnection instance.</td>
</tr>
<tr>
<td>CreateConnectionStringBuilder method</td>
<td>Returns a strongly typed DbConnectionStringBuilder instance.</td>
</tr>
<tr>
<td>CreateDataAdapter method</td>
<td>Returns a strongly typed DbDataAdapter instance.</td>
</tr>
<tr>
<td>CreateDataSourceEnumerator method</td>
<td>Returns a strongly typed DbDataSourceEnumerator instance.</td>
</tr>
<tr>
<td>CreateParameter method</td>
<td>Returns a strongly typed DbParameter instance.</td>
</tr>
<tr>
<td>CreatePermission method</td>
<td>Returns a strongly-typed CodeAccessPermission instance.</td>
</tr>
</tbody>
</table>

**See also**

- “SAFactory class” on page 359

**Instance field**

Represents the singleton instance of the SAFactory class. This field is read-only.

**Syntax**

**Visual Basic**

```vbnet
Public Shared Readonly Instance As SAFactory
```

**C#**

```csharp
public const 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 `DbProviderFactories.GetFactory`. For an example, see the SAFactory description.

**Restrictions:** The SAFactory class is not available in the .NET Compact Framework 2.0.

**See also**

- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SAFactory class” on page 359
CanCreateDataSourceEnumerator property

Always returns true, which indicates that an SADataSourceEnumerator object can be created.

Syntax

Visual Basic
Public Overrides Readonly Property CanCreateDataSourceEnumerator As Boolean

C#
public override bool CanCreateDataSourceEnumerator { get; }

Property value

A new SACommand object typed as DbCommand.

See also

- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SADataSourceEnumerator class” on page 341
- “SACommand class” on page 206

CreateCommand method

Returns a strongly typed DbCommand instance.

Syntax

Visual Basic
Public Overrides Function CreateCommand() As DbCommand

C#
public override DbCommand CreateCommand();

Return value

A new SACommand object typed as DbCommand.

See also

- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SACommand class” on page 206

CreateCommandBuilder method

Returns a strongly typed DbCommandBuilder instance.
Syntax

Visual Basic

Public Overrides Function CreateCommandBuilder() As DbCommandBuilder

C#

public override DbCommandBuilder CreateCommandBuilder();

Return value

A new SACommand object typed as DbCommand.

See also

- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SACommand class” on page 206

CreateConnection method

Returns a strongly typed DbConnection instance.

Syntax

Visual Basic

Public Overrides Function CreateConnection() As DbConnection

C#

public override DbConnection CreateConnection();

Return value

A new SACommand object typed as DbCommand.

See also

- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SACommand class” on page 206

CreateConnectionStringBuilder method

Returns a strongly typed DbConnectionStringBuilder instance.

Syntax

Visual Basic

Public Overrides Function CreateConnectionStringBuilder() As DbConnectionStringBuilder

C#

public override DbConnectionStringBuilder CreateConnectionStringBuilder();
C#

public override DbConnectionStringBuilder CreateConnectionStringBuilder();

Return value
A new SACommand object typed as DbCommand.

See also
● “SAFactory class” on page 359
● “SAFactory members” on page 360
● “SACommand class” on page 206

CreateDataAdapter method

Returns a strongly typed DbDataAdapter instance.

Syntax
Visual Basic
Public Overrides Function CreateDataAdapter() As DbDataAdapter

C#

public override DbDataAdapter CreateDataAdapter();

Return value
A new SACommand object typed as DbCommand.

See also
● “SAFactory class” on page 359
● “SAFactory members” on page 360
● “SACommand class” on page 206

CreateDataSourceEnumerator method

Returns a strongly typed DbDataSourceEnumerator instance.

Syntax
Visual Basic
Public Overrides Function CreateDataSourceEnumerator() As DbDataSourceEnumerator

C#

public override DbDataSourceEnumerator CreateDataSourceEnumerator();
Return value
A new SACommand object typed as DbCommand.

See also
- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SACommand class” on page 206

CreateParameter method
Returns a strongly typed DbParameter instance.

Syntax
Visual Basic
Public Overrides Function CreateParameter() As DbParameter

C#
public override DbParameter CreateParameter();

Return value
A new SACommand object typed as DbCommand.

See also
- “SAFactory class” on page 359
- “SAFactory members” on page 360
- “SACommand class” on page 206

CreatePermission method
Returns a strongly-typed CodeAccessPermission instance.

Syntax
Visual Basic
Public Overrides Function CreatePermission( _
    ByVal state As PermissionState _
) As CodeAccessPermission

C#
public override CodeAccessPermission CreatePermission( _
    PermissionState state _
);
Parameters
  ● state  A member of the PermissionState enumeration.

Return value
  A new SACommand object typed as DbCommand.

See also
  ● “SAFactory class” on page 359
  ● “SAFactory members” on page 360
  ● “SACommand class” on page 206

SAInfoMessageEventArgs class

Provides data for the InfoMessage event. This class cannot be inherited.

Syntax

Visual Basic
  Public NotInheritable Class SAInfoMessageEventArgs
  Inherits EventArgs

C#
  public sealed class SAInfoMessageEventArgs : EventArgs

Remarks
  There is no constructor for SAInfoMessageEventArgs.

See also
  ● “SAInfoMessageEventArgs members” on page 366

SAInfoMessageEventArgs members

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors property</td>
<td>Returns the collection of messages sent from the data source.</td>
</tr>
<tr>
<td>Message property</td>
<td>Returns the full text of the error sent from the data source.</td>
</tr>
<tr>
<td>MessageType property</td>
<td>Returns the type of the message. This can be one of: Action, Info, Status, or Warning.</td>
</tr>
<tr>
<td>NativeError property</td>
<td>Returns the SQL code returned by the database.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Source property</td>
<td>Returns the name of the SQL Anywhere .NET Data Provider.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToString method</td>
<td>Retrieves a string representation of the InfoMessage event.</td>
</tr>
</tbody>
</table>

**See also**

- “SAInfoMessageEventArgs class” on page 366

**Errors property**

Returns the collection of messages sent from the data source.

**Syntax**

Visual Basic

```vbnet
Public Readonly Property Errors As SAErroCollection
```

C#

```csharp
public SAErroCollection Errors { get; }
```

**See also**

- “SAInfoMessageEventArgs class” on page 366
- “SAInfoMessageEventArgs members” on page 366

**Message property**

Returns the full text of the error sent from the data source.

**Syntax**

Visual Basic

```vbnet
Public Readonly Property Message As String
```

C#

```csharp
public string Message { get; }
```
See also
  ● “SAInfoMessageEventArgs class” on page 366
  ● “SAInfoMessageEventArgs members” on page 366

**MessageType property**

Returns the type of the message. This can be one of: Action, Info, Status, or Warning.

**Syntax**

**Visual Basic**

Public Readonly Property **MessageType** As SAMessageType

**C#**

public SAMessageType **MessageType** { get; }

See also
  ● “SAInfoMessageEventArgs class” on page 366
  ● “SAInfoMessageEventArgs members” on page 366

**NativeError property**

Returns the SQL code returned by the database.

**Syntax**

**Visual Basic**

Public Readonly Property **NativeError** As Integer

**C#**

public int **NativeError** { get; }

See also
  ● “SAInfoMessageEventArgs class” on page 366
  ● “SAInfoMessageEventArgs members” on page 366

**Source property**

Returns the name of the SQL Anywhere .NET Data Provider.
Syntax

**Visual Basic**

Public Readonly Property **Source** As String

**C#**

public string **Source** { get; }

**See also**

- “SAInfoMessageEventArgs class” on page 366
- “SAInfoMessageEventArgs members” on page 366

### ToString method

Retrieves a string representation of the InfoMessage event.

**Syntax**

**Visual Basic**

Public Overrides Function **ToString()** As String

**C#**

public override string **ToString();**

**Return value**

A string representing the InfoMessage event.

**See also**

- “SAInfoMessageEventArgs class” on page 366
- “SAInfoMessageEventArgs members” on page 366

### SAInfoMessageEventHandler delegate

Represents the method that handles the SAConnection.InfoMessage event of an SAConnection object.

**Syntax**

**Visual Basic**

Public Delegate Sub **SAInfoMessageEventHandler**(
    ByVal **obj** As Object,
    ByVal **args** As SAInfoMessageEventArgs)

**C#**

iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
public delegate void SAInfoMessageEventHandler(
    object obj,
    SAInfoMessageEventArgs args
);

See also
● “SAConnection class” on page 246
● “InfoMessage event” on page 267

SAIsolationLevel enumeration

Specifies SQL Anywhere isolation levels. This class augments the IsolationLevel.

Syntax
Visual Basic
Public Enum SAIsolationLevel

C#
public enum SAIsolationLevel

Remarks
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].

Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaos</td>
<td>This isolation level is unsupported.</td>
<td>16</td>
</tr>
<tr>
<td>ReadCommitted</td>
<td>Sets the behavior to be equivalent to isolation level 1.</td>
<td>4096</td>
</tr>
<tr>
<td>ReadOnlySnapshot</td>
<td>For read-only statements, use a snapshot of committed data from the time when the first row is read from the database.</td>
<td>16777217</td>
</tr>
<tr>
<td>ReadUncommitted</td>
<td>Sets the behavior to be equivalent to isolation level 0.</td>
<td>256</td>
</tr>
</tbody>
</table>
### SAMessageType enumeration

Identifies the type of message. This can be one of: Action, Info, Status, or Warning.

#### Syntax

**Visual Basic**

```vbnet
Public Enum SAMessageType
```

**C#**

```csharp
public enum SAMessageType
```

#### Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Message of type ACTION.</td>
<td>2</td>
</tr>
<tr>
<td>Info</td>
<td>Message of type INFO.</td>
<td>0</td>
</tr>
<tr>
<td>Status</td>
<td>Message of type STATUS.</td>
<td>3</td>
</tr>
<tr>
<td>Warning</td>
<td>Message of type WARNING.</td>
<td>1</td>
</tr>
</tbody>
</table>
SAMetaDataCollectionNames class

Provides a list of constants for use with the SAConnection.GetSchema(String,String[]) method to retrieve metadata collections. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SAMetaDataCollectionNames

C#

public sealed class SAMetaDataCollectionNames

Remarks

This field is constant and read-only.

See also

- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

SAMetaDataCollectionNames members

Public fields

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Columns collection. This field is read-only.</td>
</tr>
<tr>
<td>DataSourceInformation field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the DataSourceInformation collection. This field is read-only.</td>
</tr>
<tr>
<td>DataTypes field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the DataTypes collection. This field is read-only.</td>
</tr>
<tr>
<td>ForeignKeys field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ForeignKeys collection. This field is read-only.</td>
</tr>
<tr>
<td>IndexColumns field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the IndexColumns collection. This field is read-only.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Indexes field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Indexes collection. This field is read-only.</td>
</tr>
<tr>
<td>MetaDataCollections field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the MetaDataCollections collection. This field is read-only.</td>
</tr>
<tr>
<td>ProcedureParameters field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ProcedureParameters collection. This field is read-only.</td>
</tr>
<tr>
<td>Procedures field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Procedures collection. This field is read-only.</td>
</tr>
<tr>
<td>ReservedWords field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ReservedWords collection. This field is read-only.</td>
</tr>
<tr>
<td>Restrictions field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Restrictions collection. This field is read-only.</td>
</tr>
<tr>
<td>Tables field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Tables collection. This field is read-only.</td>
</tr>
<tr>
<td>UserDefinedTypes field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the UserDefinedTypes collection. This field is read-only.</td>
</tr>
<tr>
<td>Users field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Users collection. This field is read-only.</td>
</tr>
<tr>
<td>ViewColumns field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ViewColumns collection. This field is read-only.</td>
</tr>
<tr>
<td>Views field</td>
<td>Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Views collection. This field is read-only.</td>
</tr>
</tbody>
</table>
See also
- “SAMetaDataCollectionNames class” on page 372
- “GetSchema(String, String[]) method” on page 262

Columns field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Columns collection. This field is read-only.

Syntax
Visual Basic
Public Shared Readonly Columns As String

C#
public const string Columns;

Example
The following code fills a DataTable with the Columns collection.

    DataTable schema = GetSchema( SAMetaDataCollectionNames.Columns );

See also
- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

DataSourceInformation field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the DataSourceInformation collection. This field is read-only.

Syntax
Visual Basic
Public Shared Readonly DataSourceInformation As String

C#
public const string DataSourceInformation;

Example
The following code fills a DataTable with the DataSourceInformation collection.

    DataTable schema = GetSchema( SAMetaDataCollectionNames.DataSourceInformation );
See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**DataTypes field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the DataTypes collection. This field is read-only.

**Syntax**

**Visual Basic**

```vbnet
Public Shared Readonly DataTypes As String
```

**C#**

```csharp
public const string DataTypes;
```

**Example**

The following code fills a DataTable with the DataTypes collection.

```vbnet
dataTable schema = GetSchema( SAMetaDataCollectionNames.DataTypes );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**ForeignKeys field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ForeignKeys collection. This field is read-only.

**Syntax**

**Visual Basic**

```vbnet
Public Shared Readonly ForeignKeys As String
```

**C#**

```csharp
public const string ForeignKeys;
```

**Example**

The following code fills a DataTable with the ForeignKeys collection.

```vbnet
dataTable schema = GetSchema( SAMetaDataCollectionNames.ForeignKeys );
```
IndexColumns field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the IndexColumns collection. This field is read-only.

Syntax

Visual Basic

Public Shared Readonly IndexColumns As String

C#

public const string IndexColumns ;

Example

The following code fills a DataTable with the IndexColumns collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.IndexColumns );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

Indexes field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Indexes collection. This field is read-only.

Syntax

Visual Basic

Public Shared Readonly Indexes As String

C#

public const string Indexes ;

Example

The following code fills a DataTable with the Indexes collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.Indexes );
```
See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**MetaDataCollections field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the MetaDataCollections collection. This field is read-only.

**Syntax**

**Visual Basic**

```vbnet
Public Shared Readonly MetaDataCollections As String
```

**C#**

```csharp
public const string MetaDataCollections;
```

**Example**

The following code fills a DataTable with the MetaDataCollections collection.

```csharp
DataTable schema = GetSchema(SAMetaDataCollectionNames.MetaDataCollections);
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**ProcedureParameters field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ProcedureParameters collection. This field is read-only.

**Syntax**

**Visual Basic**

```vbnet
Public Shared Readonly ProcedureParameters As String
```

**C#**

```csharp
public const string ProcedureParameters;
```

**Example**

The following code fills a DataTable with the ProcedureParameters collection.
DataTable schema =
GetSchema( SAMetaDataCollectionNames.ProcedureParameters );

See also
- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

Procedures field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Procedures collection. This field is read-only.

Syntax
Visual Basic

Public Shared Readonly Procedures As String

C#

public const string Procedures ;

Example

The following code fills a DataTable with the Procedures collection.

DataTable schema = GetSchema( SAMetaDataCollectionNames.Procedures );

See also
- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

ReservedWords field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ReservedWords collection. This field is read-only.

Syntax
Visual Basic

Public Shared Readonly ReservedWords As String

C#

public const string ReservedWords ;
Example

The following code fills a DataTable with the ReservedWords collection.

```csharp
DataTable schema = GetSchema(SAMetaDataCollectionNames.ReservedWords);
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

Restrictions field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Restrictions collection. This field is read-only.

Syntax

Visual Basic

```vbnet
Public Shared Readonly Restrictions As String
```

C#

```csharp
public const string Restrictions;
```

Example

The following code fills a DataTable with the Restrictions collection.

```csharp
DataTable schema = GetSchema(SAMetaDataCollectionNames.Restrictions);
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

Tables field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Tables collection. This field is read-only.

Syntax

Visual Basic

```vbnet
Public Shared Readonly Tables As String
```

C#

```csharp
public const string Tables;
```
Example

The following code fills a DataTable with the Tables collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.Tables );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

UserDefinedTypes field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the UserDefinedTypes collection. This field is read-only.

Syntax

**Visual Basic**

```vbnet
Public Shared Readonly UserDefinedTypes As String
```

**C#**

```csharp
public const string UserDefinedTypes;
```

Example

The following code fills a DataTable with the Users collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.UserDefinedTypes );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

Users field

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Users collection. This field is read-only.

Syntax

**Visual Basic**

```vbnet
Public Shared Readonly Users As String
```

**C#**

```csharp
public const string Users;
```
Example

The following code fills a DataTable with the Users collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.Users );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**ViewColumns field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the ViewColumns collection. This field is read-only.

Syntax

**Visual Basic**

```csharp
Public Shared Readonly ViewColumns As String
```

**C#**

```csharp
public const string ViewColumns;
```

Example

The following code fills a DataTable with the ViewColumns collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.ViewColumns );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**Views field**

Provides a constant for use with the SAConnection.GetSchema(String,String[]) method that represents the Views collection. This field is read-only.

Syntax

**Visual Basic**

```csharp
Public Shared Readonly Views As String
```

**C#**

```csharp
public const string Views;
```
Example

The following code fills a DataTable with the Views collection.

```csharp
DataTable schema = GetSchema( SAMetaDataCollectionNames.Views );
```

See also

- “SAMetaDataCollectionNames class” on page 372
- “SAMetaDataCollectionNames members” on page 372
- “GetSchema(String, String[]) method” on page 262

**SAParameter class**

Represents a parameter to an SACommand, and optionally, its mapping to a DataSet column. This class cannot be inherited.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SAParameter
    Inherits DbParameter
    Implements ICloneable

C#

```csharp
public sealed class SAParameter : DbParameter, ICloneable
```

**Remarks**

Implements: IDbDataParameter, IDataParameter, ICloneable

See also

- “SAParameter members” on page 382

**SAParameter members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAParameter constructors</strong></td>
<td>Initializes a new instance of the “SAParameter class” on page 382.</td>
</tr>
</tbody>
</table>
## Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbType property</td>
<td>Gets and sets the DbType of the parameter.</td>
</tr>
<tr>
<td>Direction property</td>
<td>Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.</td>
</tr>
<tr>
<td>IsNullable property</td>
<td>Gets and sets a value indicating whether the parameter accepts null values.</td>
</tr>
<tr>
<td>Offset property</td>
<td>Gets and sets the offset to the Value property.</td>
</tr>
<tr>
<td>ParameterName property</td>
<td>Gets and sets the name of the SAParameter.</td>
</tr>
<tr>
<td>Precision property</td>
<td>Gets and sets the maximum number of digits used to represent the Value property.</td>
</tr>
<tr>
<td>SADbType property</td>
<td>The SADbType of the parameter.</td>
</tr>
<tr>
<td>Scale property</td>
<td>Gets and sets the number of decimal places to which Value is resolved.</td>
</tr>
<tr>
<td>Size property</td>
<td>Gets and sets the maximum size, in bytes, of the data within the column.</td>
</tr>
<tr>
<td>SourceColumn property</td>
<td>Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.</td>
</tr>
<tr>
<td>SourceColumnNullMapping property</td>
<td>Gets and sets value that indicates whether the source column is nullable. This allows SACommandBuilder to generate Update statements for nullable columns correctly.</td>
</tr>
<tr>
<td>SourceVersion property</td>
<td>Gets and sets the DataRowVersion to use when loading Value.</td>
</tr>
<tr>
<td>Value property</td>
<td>Gets and sets the value of the parameter.</td>
</tr>
</tbody>
</table>

## Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetDbType method</td>
<td>Resets the type (the values of DbType and SADbType) associated with this SAParameter.</td>
</tr>
<tr>
<td>ToString method</td>
<td>Returns a string containing the ParameterName.</td>
</tr>
</tbody>
</table>

## See also
- “SAParameter class” on page 382
**SAPParameter constructors**

Initializes a new instance of the “SAPParameter class” on page 382.

**SAPParameter() constructor**

Initializes an SAPParameter object with null (Nothing in Visual Basic) as its value.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New()
```

**C#**

```csharp
public SAPParameter();
```

**See also**

- “SAPParameter class” on page 382
- “SAPParameter members” on page 382
- “SAPParameter constructors” on page 384

**SAPParameter(String, Object) constructor**

Initializes an SAPParameter object with the specified parameter name and value. This constructor is not recommended; it is provided for compatibility with other data providers.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New(  
    ByVal parameterName As String,  
    ByVal value As Object  
)
```

**C#**

```csharp
public SAPParameter(  
    string parameterName,  
    object value
);  
```

**Parameters**

- **parameterName**  The name of the parameter.
- **value**  An Object that is the value of the parameter.
SAParameter(String, SADbType) constructor

Initializes an SAParameter object with the specified parameter name and data type.

Syntax

**Visual Basic**

```vbnet
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADbType  
)
```

**C#**

```csharp
public SAParameter(  
    string parameterName,  
    SADbType dbType  
);
```

Parameters

- **parameterName**  The name of the parameter.
- **dbType**  One of the SADbType values.

See also

- “SAParameter class” on page 382
- “SAParameter members” on page 382
- “SAParameter constructors” on page 384

SAParameter(String, SADbType, Int32) constructor

Initializes an SAParameter object with the specified parameter name and data type.

Syntax

**Visual Basic**

```vbnet
Public Sub New(  
    ByVal parameterName As String,  
    ByVal dbType As SADbType,  
    ByVal size As Integer  
)
```
C#

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.

See also
● “SAParameter class” on page 382
● “SAParameter members” on page 382
● “SAParameter constructors” on page 384

SAParameter(String, SADbType, Int32, String) constructor

Initializes an SAParameter object with the specified parameter name, data type, and length.

Syntax
Visual Basic
Public Sub New(_
    ByVal parameterName As String, _
    ByVal dbType As SADbType, _
    ByVal size As Integer, _
    ByVal sourceColumn As String _
)

C#

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.
See also

- “SAParameter class” on page 382
- “SAParameter members” on page 382
- “SAParameter constructors” on page 384

**SAParameter(String, SADbType, Int32, ParameterDirection, Boolean, 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.

**Syntax**

**Visual Basic**

```vbnet
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#**

```csharp
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.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382
- “SAParameter constructors” on page 384

### DbType property

Gets and sets the DbType of the parameter.

**Syntax**

**Visual Basic**

Public Overrides Property DbType As DbType

**C#**

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.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382

### Direction property

Gets and sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

**Syntax**

**Visual Basic**

Public Overrides Property Direction As ParameterDirection
C#

public override ParameterDirection Direction { get; set; }

Property value
One of the ParameterDirection values.

Remarks
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.

See also
- “SAParameter class” on page 382
- “SAParameter members” on page 382

IsNullable property

Gets and sets a value indicating whether the parameter accepts null values.

Syntax
Visual Basic
Public Overrides Property IsNullable As Boolean

C#

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.

See also
- “SAParameter class” on page 382
- “SAParameter members” on page 382

Offset property

Gets and sets the offset to the Value property.

Syntax
Visual Basic
Public Property Offset As Integer
C#

public int Offset { get; set; }

Property value
The offset to the value. The default is 0.

See also
- “SAParameter class” on page 382
- “SAParameter members” on page 382

ParameterName property

Gets and sets the name of the SAParameter.

Syntax
Visual Basic

Public Overrides Property ParameterName As String

C#

class override string ParameterName { get; set; }

Property value
The default is an empty string.

Remarks
The SQL Anywhere .NET Data Provider uses positional parameters that are marked with a question mark (?) instead of named parameters.

See also
- “SAParameter class” on page 382
- “SAParameter members” on page 382

Precision property

Gets and sets the maximum number of digits used to represent the Value property.

Syntax
Visual Basic

Public Property Precision As Byte
C#

public byte Precision { get; set; }

**Property value**
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.

**Remarks**
The Precision property is only used for decimal and numeric input parameters.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382

---

**SADbType property**
The SADbType of the parameter.

**Syntax**

**Visual Basic**

Public Property SADbType As SADbType

**C#**

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.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382

---

**Scale property**
Gets and sets the number of decimal places to which Value is resolved.

**Syntax**

**Visual Basic**

Public Property Scale As Byte
C#

public byte Scale { get; set; }

**Property value**
The number of decimal places to which Value is resolved. The default is 0.

**Remarks**
The Scale property is only used for decimal and numeric input parameters.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382

---

**Size property**

Gets and sets the maximum size, in bytes, of the data within the column.

**Syntax**

**Visual Basic**

Public Overrides Property Size As Integer

**C#**

public override int Size { get; set; }

**Property 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.

**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 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.

**See also**
- “SAParameter class” on page 382
- “SAParameter members” on page 382
SourceColumn property

Gets and sets the name of the source column mapped to the DataSet and used for loading or returning the value.

Syntax

**Visual Basic**

Public Overrides Property SourceColumn As String

**C#**

public override string SourceColumn { get; set; }

Property value

A string specifying the name of the source column mapped to the DataSet and used for loading or returning the value.

Remarks

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.

See also

- “SAParameter class” on page 382
- “SAParameter members” on page 382

SourceColumnNullMapping property

Gets and sets value that indicates whether the source column is nullable. This allows SACommandBuilder to generate Update statements for nullable columns correctly.

Syntax

**Visual Basic**

Public Overrides Property SourceColumnNullMapping As Boolean

**C#**

public override bool SourceColumnNullMapping { get; set; }

Remarks

If the source column is nullable, true is returned; otherwise, false.
SourceVersion property

Gets and sets the DataRowVersion to use when loading Value.

Syntax

Visual Basic

Public Overrides Property SourceVersion As DataRowVersion

C#

custom 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.

Syntax

Visual Basic

Public Overrides Property Value As Object

C#

custom override object Value { get; set; }

Property value

An Object that specifies the value of the parameter.

Remarks

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 SDataReader 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.

See also

- “SAParameter class” on page 382
- “SAParameter members” on page 382

**ResetDbType method**

Resets the type (the values of DbType and SADbType) associated with this SAParameter.

**Syntax**

**Visual Basic**

Public Overrides Sub ResetDbType()

**C#**

public override void ResetDbType();

See also

- “SAParameter class” on page 382
- “SAParameter members” on page 382

**ToString method**

Returns a string containing the ParameterName.

**Syntax**

**Visual Basic**

Public Overrides Function ToString() As String

**C#**

public override string ToString();

**Return value**

The name of the parameter.
See also

- "SAParameter class" on page 382
- "SAParameter members" on page 382

**SAParameterCollection class**

Represents all parameters to an SACommand object and, optionally, their mapping to a DataSet column. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SAParameterCollection
Inherits DbParameterCollection

**C#**

public sealed class SAParameterCollection : DbParameterCollection

**Remarks**

There is no constructor for SAParameterCollection. You obtain an SAParameterCollection object from the SACommand.Parameters property of an SACommand object.

See also

- "SAParameterCollection members" on page 396
- "SACommand class" on page 206
- "Parameters property" on page 213
- "SAParameter class" on page 382
- "SAParameterCollection class" on page 396

**SAParameterCollection members**

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Returns the number of SAParameter objects in the collection.</td>
</tr>
<tr>
<td>IsFixedSize property</td>
<td>Gets a value that indicates whether the SAParameterCollection has a fixed size.</td>
</tr>
<tr>
<td>IsReadOnly property</td>
<td>Gets a value that indicates whether the SAParameterCollection is read-only.</td>
</tr>
</tbody>
</table>
### Member name

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsSynchronized property</td>
<td>Gets a value that indicates whether the SAParameterCollection object is synchronized.</td>
</tr>
<tr>
<td>Item properties</td>
<td>Gets and sets the SAParameter object at the specified index.</td>
</tr>
<tr>
<td>SyncRoot property</td>
<td>Gets an object that can be used to synchronize access to the SAParameterCollection.</td>
</tr>
</tbody>
</table>

### Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add methods</td>
<td>Adds an SAParameter object to this collection.</td>
</tr>
<tr>
<td>AddRange methods</td>
<td>Adds an array of values to the end of the SAParameterCollection.</td>
</tr>
<tr>
<td>AddWithValue method</td>
<td>Adds a value to the end of this collection.</td>
</tr>
<tr>
<td>Clear method</td>
<td>Removes all items from the collection.</td>
</tr>
<tr>
<td>Contains methods</td>
<td>Indicates whether an SAParameter object exists in the collection.</td>
</tr>
<tr>
<td>CopyTo method</td>
<td>Copies SAParameter objects from the SAParameterCollection to the specified array.</td>
</tr>
<tr>
<td>GetEnumerator method</td>
<td>Returns an enumerator that iterates through the SAParameterCollection.</td>
</tr>
<tr>
<td>IndexOf methods</td>
<td>Returns the location of the SAParameter object in the collection.</td>
</tr>
<tr>
<td>Insert method</td>
<td>Inserts an SAParameter object in the collection at the specified index.</td>
</tr>
<tr>
<td>Remove method</td>
<td>Removes the specified SAParameter object from the collection.</td>
</tr>
<tr>
<td>RemoveAt methods</td>
<td>Removes the specified SAParameter object from the collection.</td>
</tr>
</tbody>
</table>

### See also

- “SAParameterCollection class” on page 396
- “SACommand class” on page 206
- “Parameters property” on page 213
- “SAParameter class” on page 382
- “SAParameterCollection class” on page 396
Count property

Returns the number of SAParameter objects in the collection.

Syntax

Visual Basic

Public Overrides Readonly Property Count As Integer

C#

public override int Count { get; }

Property value

The number of SAParameter objects in the collection.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “SAParameter class” on page 382
- “SAParameterCollection class” on page 396

IsFixedSize property

Gets a value that indicates whether the SAParameterCollection has a fixed size.

Syntax

Visual Basic

Public Overrides Readonly Property IsFixedSize As Boolean

C#

public override bool IsFixedSize { get; }

Property value

True if this collection has a fixed size, false otherwise.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396

IsReadOnly property

Gets a value that indicates whether the SAParameterCollection is read-only.
**IsReadOnly** property

Gets a value that indicates whether the SAParameterCollection object is read-only.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property **IsReadOnly** As Boolean

**C#**

public override bool **IsReadOnly** { get; }

**Property value**

True if this collection is read-only, false otherwise.

**See also**

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396

**IsSynchronized** property

Gets a value that indicates whether the SAParameterCollection object is synchronized.

**Syntax**

**Visual Basic**

Public Overrides Readonly Property **IsSynchronized** As Boolean

**C#**

public override bool **IsSynchronized** { get; }

**Property value**

True if this collection is synchronized, false otherwise.

**See also**

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396

**Item properties**

Gets and sets the SAParameter object at the specified index.

**Item(Int32)** property

Gets and sets the SAParameter object at the specified index.
Syntax

Visual Basic

Public Property Item ( _
    ByVal index As Integer _
) As SAParameter

C#

public SAParameter this [ 
    int index
] { get; set; }

Parameters

- index The zero-based index of the parameter to retrieve.

Property value

The SAParameter at the specified index.

Remarks

In C#, this property is the indexer for the SAParameterCollection object.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Item properties” on page 399
- “SAParameter class” on page 382
- “SAParameterCollection class” on page 396

Item(String) property

Gets and sets the SAParameter object at the specified index.

Syntax

Visual Basic

Public Property Item ( _
    ByVal parameterName As String _
) As SAParameter

C#

public SAParameter this [ 
    string parameterName
] { get; set; }

Parameters

- parameterName The name of the parameter to retrieve.
Property value

The SAParameter object with the specified name.

Remarks

In C#, this property is the indexer for the SAParameterCollection object.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Item properties” on page 399
- “SAParameter class” on page 382
- “SAParameterCollection class” on page 396
- “Item(Int32) property” on page 315
- “GetOrdinal method” on page 330
- “GetValue(Int32) method” on page 337
- “GetFieldType method” on page 325

SyncRoot property

Gets an object that can be used to synchronize access to the SAParameterCollection.

Syntax

Visual Basic

Public Overrides Readonly Property SyncRoot As Object

C#

public override object SyncRoot { get; }

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396

Add methods

Adds an SAParameter object to this collection.

Add(Object) method

Adds an SAParameter object to this collection.
Syntax

Visual Basic

Public Overrides Function Add(_
    ByVal value As Object _
) As Integer

C#

public override int Add(
    object value
);

Parameters

- value The SAParameter object to add to the collection.

Return value

The index of the new SAParameter object.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Add methods” on page 401
- “SAParameter class” on page 382

Add(SAParameter) method

Adds an SAParameter object to this collection.

Syntax

Visual Basic

Public Function Add(_
    ByVal value As SAParameter _
) As SAParameter

C#

public SAParameter Add(
    SAParameter value
);

Parameters

- value The SAParameter object to add to the collection.

Return value

The new SAParameter object.
See also
- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Add methods” on page 401

Add(String, Object) method

Adds an SAParameter object to this collection, created using the specified parameter name and value, to the collection.

Syntax

Visual Basic

Public Function Add(  
    ByVal paramName As String, _
    ByVal value As Object _
) As SAParameter

C#

public SAParameter Add(  
    string paramName,  
    object value  
);  

Parameters

- paramName The name of the parameter.
- value The value of the parameter to add to the connection.

Return value

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
- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Add methods” on page 401
- “SAParameter class” on page 382

Add(String, SADbType) method

Adds an SAParameter object to this collection, created using the specified parameter name and data type, to the collection.
Syntax

Visual Basic

Public Function Add(_
    ByVal parameterName As String, _
    ByVal saDbType As SADbType _
) As SAParameter

C#

public SAParameter Add(
    string parameterName,
    SADbType saDbType
);

Parameters

- **parameterName**  The name of the parameter.
- **saDbType**  One of the SADbType values.

Return value

The new SAParameter object.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Add methods” on page 401
- “SADbType enumeration” on page 343
- “Add(SAParameter) method” on page 402
- “Add(String, Object) method” on page 403

Add(String, SADbType, Int32) method

Adds an SAParameter object to this collection, created using the specified parameter name, data type, and length, to the collection.

Syntax

Visual Basic

Public Function Add(_
    ByVal parameterName As String, _
    ByVal saDbType As SADbType, _
    ByVal size As Integer _
) As SAParameter

C#

public SAParameter 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.

Return value
The new SAParameter object.

See also
● “SAParameterCollection class” on page 396
● “SAParameterCollection members” on page 396
● “Add methods” on page 401
● “SADbType enumeration” on page 343
● “Add(SAParameter) method” on page 402
● “Add(String, Object) method” on page 403

Add(String, SADbType, Int32, String) method
Adds an SAParameter object to this collection, created using the specified parameter name, data type, length, and source column name, to the collection.

Syntax
Visual Basic
Public Function Add( _
    ByVal parameterName As String, _
    ByVal saDbType As SADbType, _
    ByVal size As Integer, _
    ByVal sourceColumn As String _
) As SAParameter

C#

public SAParameter 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.

Return value
The new SAParameter object.

See also
- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Add methods” on page 401
- “SADbType enumeration” on page 343
- “Add(SAParameter) method” on page 402
- “Add(String, Object) method” on page 403

AddRange methods
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.

Syntax
Visual Basic
Public Overrides Sub AddRange( _
   ByVal values As Array _
)

C#
public override void AddRange(
   Array values
);

Parameters
- values  The values to add.

See also
- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “AddRange methods” on page 406
**AddRange(SAParameter[]) method**

Adds an array of values to the end of the SAParameterCollection.

**Syntax**

**Visual Basic**

Public Sub AddRange(_
    ByVal values As SAParameter() _
) _

**C#**

public void AddRange(_
    SAParameter[] values _
);

**Parameters**

- **values**  An array of SAParameter objects to add to the end of this collection.

**See also**

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “AddRange methods” on page 406

**AddWithValue method**

Adds a value to the end of this collection.

**Syntax**

**Visual Basic**

Public Function AddWithValue(_
    ByVal parameterName As String, _
    ByVal value As Object _
) As SAParameter _

**C#**

public SAParameter AddWithValue(_
    string parameterName, _
    object value _
);

**Parameters**

- **parameterName**  The name of the parameter.
- **value**  The value to be added.
Return value

The new SAParameter object.

See also

● “SAParameterCollection class” on page 396
● “SAParameterCollection members” on page 396

Clear method

Removes all items from the collection.

Syntax

Visual Basic

Public Overrides Sub Clear()

C#

cpyublic override void Clear();

See also

● “SAParameterCollection class” on page 396
● “SAParameterCollection members” on page 396

Contains methods

Indicates whether an SAParameter object exists in the collection.

Contains(Object) method

Indicates whether an SAParameter object exists in the collection.

Syntax

Visual Basic

Public Overrides Function Contains( _
    ByVal value As Object _
  ) As Boolean

C#

public override bool Contains(
    object value
);
Parameters

- **value** The SAParameter object to find.

Return value

True if the collection contains the SAParameter object. Otherwise, false.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Contains methods” on page 408
- “SAParameter class” on page 382
- “Contains(String) method” on page 409

Contains(String) method

Indicates whether an SAParameter object exists in the collection.

Syntax

**Visual Basic**

```vbnet
Public Overrides Function Contains( 
  ByVal value As String 
) As Boolean
```

**C#**

```csharp
public override bool Contains(
  string value
);
```

Parameters

- **value** The name of the parameter to search for.

Return value

True if the collection contains the SAParameter object. Otherwise, false.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “Contains methods” on page 408
- “SAParameter class” on page 382
- “Contains(Object) method” on page 408
**CopyTo method**

Copies SAPParameter objects from the SAParameterCollection to the specified array.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Sub CopyTo(
    ByVal array As Array,
    ByVal index As Integer
)
```

**C#**

```csharp
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**

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “SAParameter class” on page 382
- “SAParameterCollection class” on page 396

**GetEnumerator method**

Returns an enumerator that iterates through the SAParameterCollection.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Function GetEnumerator() As IEnumerator
```

**C#**

```csharp
public override IEnumerator GetEnumerator();
```

**Return value**

An `IEnumerator` for the SAParameterCollection object.
IndexOf methods

Returns the location of the SAParameter object in the collection.

IndexOf(Object) method

Returns the location of the SAParameter object in the collection.

Syntax

Visual Basic
Public Overrides Function IndexOf( _
   ByVal value As Object _
) As Integer

C#

public override int IndexOf(  
   object value
 );

Parameters

- **value**  The SAParameter object to locate.

Return value

The zero-based location of the SAParameter object in the collection.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “SAParameterCollection class” on page 396

IndexOf(String) method

Returns the location of the SAParameter object in the collection.
Syntax

Visual Basic

Public Overrides Function IndexOf( _
    ByVal parameterName As String _
) As Integer

C#

public override int IndexOf(
    string parameterName
);

Parameters

- **parameterName**  The name of the parameter to locate.

Return value

The zero-based index of the SAParameter object in the collection.

See also

- “SAParameterCollection class” on page 396
- “SAParameterCollection members” on page 396
- “IndexOf methods” on page 411
- “SAParameter class” on page 382
- “IndexOf(Object) method” on page 411

**Insert method**

Inserts an SAParameter object in the collection at the specified index.

Syntax

Visual Basic

Public Overrides Sub Insert( _
    ByVal index As Integer, _
    ByVal value As Object _
)

C#

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.
See also
  ● “SAParameterCollection class” on page 396
  ● “SAParameterCollection members” on page 396

Remove method

Removes the specified SAParameter object from the collection.

Syntax
  Visual Basic
  Public Overrides Sub Remove( 
  ByVal value As Object 
  )

  C#
  public override void Remove(
  object value
  );

Parameters
  ● value   The SAParameter object to remove from the collection.

See also
  ● “SAParameterCollection class” on page 396
  ● “SAParameterCollection members” on page 396

RemoveAt methods

Removes the specified SAParameter object from the collection.

RemoveAt(Int32) method

Removes the specified SAParameter object from the collection.

Syntax
  Visual Basic
  Public Overrides Sub RemoveAt( 
  ByVal index As Integer 
  )

  C#
  public override void RemoveAt(
int index
);

Parameters
● index The zero-based index of the parameter to remove.

See also
● “SAParameterCollection class” on page 396
● “SAParameterCollection members” on page 396
● “RemoveAt methods” on page 413
● “RemoveAt(Int32) method” on page 413

RemoveAt(String) method

Removes the specified SAParameter object from the collection.

Syntax
Visual Basic
Public Overrides Sub RemoveAt( _
   ByVal parameterName As String _
)

C#
public override void RemoveAt( _
   string parameterName _
);

Parameters
● parameterName The name of the SAParameter object to remove.

See also
● “SAParameterCollection class” on page 396
● “SAParameterCollection members” on page 396
● “RemoveAt methods” on page 413
● “RemoveAt(Int32) method” on page 413

**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. This class cannot be inherited.

Syntax
Visual Basic
Public NotInheritable Class **SAPermission**  
Inherits DBDataPermission

C#  
public sealed class **SAPermission** : DBDataPermission

Remarks  
Base classes **DBDataPermission**

See also  
- “**SAPermission members**” on page 415

**SAPermission members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAPermission constructor</strong></td>
<td>Initializes a new instance of the SAPermission class.</td>
</tr>
</tbody>
</table>

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AllowBlankPassword</strong> (inherited from DBDataPermission)</td>
<td>Gets a value indicating whether a blank password is allowed.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong> (inherited from DBDataPermission)</td>
<td>Adds access for the specified connection string to the existing state of the DBDataPermission.</td>
</tr>
<tr>
<td><strong>Assert</strong> (inherited from CodeAccessPermission)</td>
<td>Declares that the calling code can access the resource protected by a permission demand through the code that calls this method, even if callers higher in the stack have not been granted permission to access the resource. Using CodeAccessPermission.Assert can create security issues.</td>
</tr>
<tr>
<td><strong>Copy</strong> (inherited from DBDataPermission)</td>
<td>Creates and returns an identical copy of the current permission object.</td>
</tr>
<tr>
<td><strong>Demand</strong> (inherited from CodeAccessPermission)</td>
<td>Forces a SecurityException at run time if all callers higher in the call stack have not been granted the permission specified by the current instance.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Deny</strong> (inherited from CodeAc-</td>
<td>Prevents callers higher in the call stack from using the code that calls this method to access the resource specified by the current instance.</td>
</tr>
<tr>
<td>cessPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>Equals</strong> (inherited from CodeAc-</td>
<td>Determines whether the specified CodeAccessPermission object is equal to the current CodeAccessPermission.</td>
</tr>
<tr>
<td>cessPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>FromXml</strong> (inherited from DBDa-</td>
<td>Reconstructs a security object with a specified state from an XML encoding.</td>
</tr>
<tr>
<td>taPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>GetHashCode</strong> (inherited from</td>
<td>Gets a hash code for the CodeAccessPermission object that is suitable for use in hashing algorithms and data structures such as a hash table.</td>
</tr>
<tr>
<td>CodeAccessPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>Intersect</strong> (inherited from DBDa-</td>
<td>Returns a new permission object representing the intersection of the current permission object and the specified permission object.</td>
</tr>
<tr>
<td>taPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>IsSubsetOf</strong> (inherited from DBD-</td>
<td>Returns a value indicating whether the current permission object is a subset of the specified permission object.</td>
</tr>
<tr>
<td>ataPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>IsUnrestricted</strong> (inherited from</td>
<td>Returns a value indicating whether the permission can be represented as unrestricted without any knowledge of the permission semantics.</td>
</tr>
<tr>
<td>DBDataPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>PermitOnly</strong> (inherited from Co-</td>
<td>Prevents callers higher in the call stack from using the code that calls this method to access all resources except for the resource specified by the current instance.</td>
</tr>
<tr>
<td>deAccessPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>ToString</strong> (inherited from Co-</td>
<td>Creates and returns a string representation of the current permission object.</td>
</tr>
<tr>
<td>deAccessPermission)</td>
<td></td>
</tr>
<tr>
<td><strong>ToXml</strong> (inherited from DBData-</td>
<td>Creates an XML encoding of the security object and its current state.</td>
</tr>
<tr>
<td>Permission)</td>
<td></td>
</tr>
<tr>
<td><strong>Union</strong> (inherited from DBData-</td>
<td>Returns a new permission object that is the union of the current and specified permission objects.</td>
</tr>
<tr>
<td>Permission)</td>
<td></td>
</tr>
</tbody>
</table>

**See also**

- “SAPermission class” on page 414

**SAPermission constructor**

Initializes a new instance of the SAPermission class.

**Syntax**

**Visual Basic**

Public Sub New( _
ByVal state As PermissionState
)

C#

public SAPermission(
    PermissionState state
);

Parameters
● state One of the PermissionState values.

See also
● “SAPermission class” on page 414
● “SAPermission members” on page 415

SAPermissionAttribute class

Associates a security action with a custom security attribute. This class cannot be inherited.

Syntax
Visual Basic
Public NotInheritable Class SAPermissionAttribute
Inherits DBDataPermissionAttribute

C#

public sealed class SAPermissionAttribute : DBDataPermissionAttribute

See also
● “SAPermissionAttribute members” on page 417

SAPermissionAttribute members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPermissionAttribute con-</td>
<td>Initializes a new instance of the SAPermissionAttribute class.</td>
</tr>
<tr>
<td>structor</td>
<td></td>
</tr>
</tbody>
</table>
### Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong> (inherited from SecurityAttribute)</td>
<td>Gets or sets a security action.</td>
</tr>
<tr>
<td><strong>AllowBlankPassword</strong> (inherited from DBDataPermissionAttribute)</td>
<td>Gets or sets a value indicating whether a blank password is allowed.</td>
</tr>
<tr>
<td><strong>ConnectionString</strong> (inherited from DBDataPermissionAttribute)</td>
<td>Gets or sets a permitted connection string.</td>
</tr>
<tr>
<td><strong>KeyRestrictionBehavior</strong> (inherited from DBDataPermissionAttribute)</td>
<td>Identifies whether the list of connection string parameters identified by the DBDataPermissionAttribute.KeyRestrictions are the only connection string parameters allowed.</td>
</tr>
<tr>
<td><strong>KeyRestrictions</strong> (inherited from DBDataPermissionAttribute)</td>
<td>Gets or sets connection string parameters that are allowed or disallowed.</td>
</tr>
<tr>
<td><strong>TypeId</strong> (inherited from Attribute)</td>
<td>When implemented in a derived class, gets a unique identifier for this Attribute.</td>
</tr>
<tr>
<td><strong>Unrestricted</strong> (inherited from SecurityAttribute)</td>
<td>Gets or sets a value indicating whether full (unrestricted) permission to the resource protected by the attribute is declared.</td>
</tr>
</tbody>
</table>

### Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CreatePermission method</strong></td>
<td>Returns an SAPermission object that is configured according to the attribute properties.</td>
</tr>
<tr>
<td><strong>Equals</strong> (inherited from Attribute)</td>
<td>Returns a value that indicates whether this instance is equal to a specified object.</td>
</tr>
<tr>
<td><strong>GetHashCode</strong> (inherited from Attribute)</td>
<td>Returns the hash code for this instance.</td>
</tr>
<tr>
<td><strong>IsDefaultAttribute</strong> (inherited from Attribute)</td>
<td>When overridden in a derived class, indicates whether the value of this instance is the default value for the derived class.</td>
</tr>
<tr>
<td><strong>Match</strong> (inherited from Attribute)</td>
<td>When overridden in a derived class, returns a value that indicates whether this instance equals a specified object.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ShouldSerializeConnectionString (inherited from DBDataPermissionAttribute)</td>
<td>Identifies whether the attribute should serialize the connection string.</td>
</tr>
<tr>
<td>ShouldSerializeKeyRestrictions (inherited from DBDataPermissionAttribute)</td>
<td>Identifies whether the attribute should serialize the set of key restrictions.</td>
</tr>
</tbody>
</table>

See also
- “SAPermissionAttribute class” on page 417

**SAPermissionAttribute constructor**

Initializes a new instance of the SAPermissionAttribute class.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New( ByVal action As SecurityAction )
```

**C#**

```csharp
public SAPermissionAttribute(SecurityAction action);
```

**Parameters**

- `action` One of the SecurityAction values representing an action that can be performed using declarative security.

See also
- “SAPermissionAttribute class” on page 417
- “SAPermissionAttribute members” on page 417

**CreatePermission method**

Returns an SAPermission object that is configured according to the attribute properties.

**Syntax**

**Visual Basic**

```vbnet
Public Overrides Function CreatePermission() As IPermission
```
C# public override IPermission CreatePermission();

See also
- “SAPermissionAttribute class” on page 417
- “SAPermissionAttribute members” on page 417

SARowsCopiedEventArgs class

Represents the set of arguments passed to the SARowsCopiedEventHandler. This class cannot be inherited.

Syntax
Visual Basic
Public NotInheritable Class SARowsCopiedEventArgs

C#
public sealed class SARowsCopiedEventArgs

Remarks
Restrictions: The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also
- “SARowsCopiedEventArgs members” on page 420

SARowsCopiedEventArgs members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SARowsCopiedEventArgs constructor</td>
<td>Creates a new instance of the SARowsCopiedEventArgs object.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort property</td>
<td>Gets or sets a value that indicates whether the bulk-copy operation should be aborted.</td>
</tr>
<tr>
<td>RowsCopied property</td>
<td>Gets the number of rows copied during the current bulk-copy operation.</td>
</tr>
</tbody>
</table>
See also

- “SARowsCopiedEventArgs class” on page 420

### SARowsCopiedEventArgs constructor

Creates a new instance of the SARowsCopiedEventArgs object.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New(
    ByVal rowsCopied As Long
)
```

**C#**

```csharp
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**

**Restrictions:** The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

See also

- “SARowsCopiedEventArgs class” on page 420
- “SARowsCopiedEventArgs members” on page 420

### Abort property

Gets or sets a value that indicates whether the bulk-copy operation should be aborted.

**Syntax**

**Visual Basic**

```vbnet
Public Property Abort As Boolean
```

**C#**

```csharp
public bool Abort { get; set; }
```

**Remarks**

**Restrictions:** The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.
See also

- “SARowsCopiedEventArgs class” on page 420
- “SARowsCopiedEventArgs members” on page 420

**RowsCopied property**

Gets the number of rows copied during the current bulk-copy operation.

**Syntax**

**Visual Basic**

```vbnet
Public Readonly Property RowsCopied As Long
```

**C#**

```csharp
public long RowsCopied { get; }
```

**Remarks**

**Restrictions:** The SARowsCopiedEventArgs class is not available in the .NET Compact Framework 2.0.

**See also**

- “SARowsCopiedEventArgs class” on page 420
- “SARowsCopiedEventArgs members” on page 420

**SARowsCopiedEventHandler delegate**

Represents the method that handles the SABulkCopy.SARowsCopied event of an SABulkCopy.

**Syntax**

**Visual Basic**

```vbnet
Public Delegate Sub SARowsCopiedEventHandler( _
    ByVal sender As Object, _
    ByVal rowsCopiedEventArgs As SARowsCopiedEventArgs _
) 
```

**C#**

```csharp
public delegate void SARowsCopiedEventHandler( 
    object sender, 
    SARowsCopiedEventArgs rowsCopiedEventArgs 
);
```

**Remarks**

**Restrictions:** The SARowsCopiedEventHandler delegate is not available in the .NET Compact Framework 2.0.
SARowUpdatedEventArgs class

Provides data for the RowUpdated event. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SARowUpdatedEventArgs
  Inherits RowUpdatedEventArgs

C#

public sealed class SARowUpdatedEventArgs : RowUpdatedEventArgs

See also

● “SARowUpdatedEventArgs members” on page 423

SARowUpdatedEventArgs members

Public constructors

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SARowUpdatedEventArgs constructor</td>
<td>Initializes a new instance of the SARowUpdatedEventArgs class.</td>
</tr>
</tbody>
</table>

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command property</td>
<td>Gets the SACommand that is executed when DataAdapter.Update is called.</td>
</tr>
<tr>
<td>Errors (inherited from RowUp-</td>
<td>Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command was executed.</td>
</tr>
<tr>
<td>datedEventArgs)</td>
<td></td>
</tr>
<tr>
<td>RecordsAffected property</td>
<td>Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.</td>
</tr>
<tr>
<td>Row (inherited from RowUpdat-</td>
<td>Gets the DataRow sent through an DbDataAdapter.Update.</td>
</tr>
<tr>
<td>edEventArgs)</td>
<td></td>
</tr>
</tbody>
</table>
### Member name

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RowCount</strong> (inherited from <strong>RowUpdatedEventArgs</strong>)</td>
<td>Gets the number of rows processed in a batch of updated records.</td>
</tr>
<tr>
<td><strong>StatementType</strong> (inherited from <strong>RowUpdatedEventArgs</strong>)</td>
<td>Gets the type of SQL statement executed.</td>
</tr>
<tr>
<td><strong>Status</strong> (inherited from <strong>RowUpdatedEventArgs</strong>)</td>
<td>Gets the <strong>UpdateStatus</strong> of the <strong>RowUpdatedEventArgs.Command</strong>.</td>
</tr>
<tr>
<td><strong>TableMapping</strong> (inherited from <strong>RowUpdatedEventArgs</strong>)</td>
<td>Gets the <strong>DataTableMapping</strong> sent through an <strong>DbDataAdapter.Update</strong>.</td>
</tr>
</tbody>
</table>

### Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CopyToRows</strong> (inherited from <strong>RowUpdatedEventArgs</strong>)</td>
<td>Copies references to the modified rows into the provided array.</td>
</tr>
</tbody>
</table>

### See also

- “**SARowUpdatedEventArgs class**” on page 423

### SARowUpdatedEventArgs constructor

Initializes a new instance of the **SARowUpdatedEventArgs** class.

#### Syntax

**Visual Basic**

```vbnet
Public Sub New(_
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
)
```

**C#**

```csharp
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.

See also

- “SARowUpdatedEventArgs class” on page 423
- “SARowUpdatedEventArgs members” on page 423

Command property

Gets the SACommand that is executed when DataAdapter.Update is called.

Syntax

**Visual Basic**

```vbnet
Public Readonly Property Command As SACommand
```

**C#**

```csharp
public SACommand Command { get; }
```

See also

- “SARowUpdatedEventArgs class” on page 423
- “SARowUpdatedEventArgs members” on page 423

RecordsAffected property

Returns the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax

**Visual Basic**

```vbnet
Public Readonly Property RecordsAffected As Integer
```

**C#**

```csharp
public int RecordsAffected { get; }
```

Property value

The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for SELECT statements.
See also
- “SARowUpdatedEventArgs class” on page 423
- “SARowUpdatedEventArgs members” on page 423

**SARowUpdatedEventArgs class**

Provides data for the RowUpdating event. This class cannot be inherited.

**Syntax**

**Visual Basic**

```vbnet
Public NotInheritable Class SARowUpdatingEventArgs
    Inherits RowUpdatingEventArgs
End Class
```

**C#**

```csharp
public sealed class SARowUpdatingEventArgs : RowUpdatingEventArgs
```

**See also**
- “SARowUpdatingEventArgs members” on page 427
**SARowUpdatingEventArgs members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SARowUpdatingEventArgs constructor</td>
<td>Initializes a new instance of the SARowUpdatingEventArgs class.</td>
</tr>
</tbody>
</table>

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command property</td>
<td>Specifies the SACommand to execute when performing the Update.</td>
</tr>
<tr>
<td>Errors (inherited from RowUpdatingEventArgs)</td>
<td>Gets any errors generated by the .NET Framework data provider when the RowUpdatedEventArgs.Command executes.</td>
</tr>
<tr>
<td>Row (inherited from RowUpdatingEventArgs)</td>
<td>Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.</td>
</tr>
<tr>
<td>StatementType (inherited from RowUpdatingEventArgs)</td>
<td>Gets the type of SQL statement to execute.</td>
</tr>
<tr>
<td>Status (inherited from RowUpdatingEventArgs)</td>
<td>Gets or sets the UpdateStatus of the RowUpdatedEventArgs.Command.</td>
</tr>
<tr>
<td>TableMapping (inherited from RowUpdatingEventArgs)</td>
<td>Gets the DataTableMapping to send through the DbDataAdapter.Update.</td>
</tr>
</tbody>
</table>

**See also**

- “SARowUpdatingEventArgs class” on page 426

### SARowUpdatingEventArgs constructor

Initializes a new instance of the SARowUpdatingEventArgs class.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New(  
    ByVal row As DataRow,  
    ByVal command As IDbCommand,  
    ByVal statementType As StatementType,  
    ByVal tableMapping As DataTableMapping  
)
```
C#

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.

See also

- “SARowUpdatingEventArgs class” on page 426
- “SARowUpdatingEventArgs members” on page 427

**Command property**

Specifies the SACommand to execute when performing the Update.

Syntax

Visual Basic

Public Property Command As SACommand

C#

public SACommand Command { get; set; }

See also

- “SARowUpdatingEventArgs class” on page 426
- “SARowUpdatingEventArgs members” on page 427

**SARowUpdatingEventHandler delegate**

Represents the method that handles the RowUpdating event of an SDataAdapter.

Syntax

Visual Basic

Public Delegate Sub SARowUpdatingEventHandler( _ 
    ByVal sender As Object, _
    ByVal eventArgs As SARowUpdatingEventArgs)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
ByVal e As SARowUpdatingEventArgs _
)

C#

public delegate void SARowUpdatingEventHandler(
    object sender,
    SARowUpdatingEventArgs e
);

**SATcpOptionsBuilder class**

Provides a simple way to create and manage the TCP options portion of connection strings used by the SAConnection object. This class cannot be inherited.

**Syntax**

**Visual Basic**

Public NotInheritable Class SATcpOptionsBuilder
    Inherits SAConnectionStringBuilderBase

**C#**

public sealed class SATcpOptionsBuilder : SAConnectionStringBuilderBase

**Remarks**

**Restrictions:** The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

**See also**

- “SATcpOptionsBuilder members” on page 429
- “SAConnection class” on page 246

**SATcpOptionsBuilder members**

**Public constructors**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SATcpOptionsBuilder constructors</td>
<td>Initializes a new instance of the “SATcpOptionsBuilder class” on page 429.</td>
</tr>
</tbody>
</table>

**Public properties**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast property</td>
<td>Gets or sets the Broadcast option.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BroadcastListener property</td>
<td>Gets or sets the BroadcastListener option.</td>
</tr>
<tr>
<td>BrowsableConnectionString</td>
<td>Gets or sets a value that indicates whether the DbConnectionStringBuilder.ConnectionString is visible in Visual Studio designers.</td>
</tr>
<tr>
<td></td>
<td>(inherited from DbConnectionStringBuilder)</td>
</tr>
<tr>
<td>ClientPort property</td>
<td>Gets or sets the ClientPort option.</td>
</tr>
<tr>
<td>ConnectionString</td>
<td>Gets or sets the connection string associated with the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td></td>
<td>(inherited from DbConnectionStringBuilder)</td>
</tr>
<tr>
<td>Count (inherited from</td>
<td>Gets the current number of keys that are contained within the DbConnectionStringBuilder.ConnectionString.</td>
</tr>
<tr>
<td>DbConnectionStringBuilder)</td>
<td></td>
</tr>
<tr>
<td>DoBroadcast property</td>
<td>Gets or sets the DoBroadcast option.</td>
</tr>
<tr>
<td>Host property</td>
<td>Gets or sets the Host option.</td>
</tr>
<tr>
<td>IPV6 property</td>
<td>Gets or sets the IPV6 option.</td>
</tr>
<tr>
<td>IsFixedSize (inherited from</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder has a fixed size.</td>
</tr>
<tr>
<td>DbConnectionStringBuilder)</td>
<td></td>
</tr>
<tr>
<td>IsReadOnly (inherited from</td>
<td>Gets a value that indicates whether the DbConnectionStringBuilder is read-only.</td>
</tr>
<tr>
<td>DbConnectionStringBuilder)</td>
<td></td>
</tr>
<tr>
<td>Item property (inherited</td>
<td>Gets or sets the value of the connection keyword.</td>
</tr>
<tr>
<td>From SAConnectionStringBuilder-Base)</td>
<td></td>
</tr>
<tr>
<td>Keys property (inherited</td>
<td>Gets an System.Collections.ICollection that contains the keys in the SAConnectionStringBuilder.</td>
</tr>
<tr>
<td>From SAConnectionStringBuilder-Base)</td>
<td></td>
</tr>
<tr>
<td>LDAP property</td>
<td>Gets or sets the LDAP option.</td>
</tr>
<tr>
<td>LocalOnly property</td>
<td>Gets or sets the LocalOnly option.</td>
</tr>
<tr>
<td>MyIP property</td>
<td>Gets or sets the MyIP option.</td>
</tr>
<tr>
<td>ReceiveBufferSize property</td>
<td>Gets or sets the ReceiveBufferSize option.</td>
</tr>
<tr>
<td>SendBufferSize property</td>
<td>Gets or sets the Send BufferSize option.</td>
</tr>
<tr>
<td>ServerPort property</td>
<td>Gets or sets the ServerPort option.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TDS property</td>
<td>Gets or sets the TDS option.</td>
</tr>
<tr>
<td>Timeout property</td>
<td>Gets or sets the Timeout option.</td>
</tr>
<tr>
<td><strong>Values</strong> (inherited from DbConnectionStringBuilder)</td>
<td>Gets an ICollection that contains the values in the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td>VerifyServerName property</td>
<td>Gets or sets the VerifyServerName option.</td>
</tr>
</tbody>
</table>

**Public methods**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (inherited from DbConnectionStringBuilder)</td>
<td>Adds an entry with the specified key and value into the DbConnectionStringBuilder.</td>
</tr>
<tr>
<td>Clear (inherited from DbConnectionStringBuilder)</td>
<td>Clears the contents of the DbConnectionStringBuilder instance.</td>
</tr>
<tr>
<td>ContainsKey method (inherited from SAConnectionStringBuilderBase)</td>
<td>Determines whether the SAConnectionStringBuilder object contains a specific keyword.</td>
</tr>
<tr>
<td>EquivalentTo (inherited from DbConnectionStringBuilder)</td>
<td>Compares the connection information in this DbConnectionStringBuilder object with the connection information in the supplied object.</td>
</tr>
<tr>
<td>GetKeyword method (inherited from SAConnectionStringBuilderBase)</td>
<td>Gets the keyword for specified SAConnectionStringBuilder property.</td>
</tr>
<tr>
<td>GetUseLongNameAsKeyword method (inherited from SAConnectionStringBuilderBase)</td>
<td>Gets a boolean values that indicates whether long connection parameter names are used in the connection string.</td>
</tr>
<tr>
<td>Remove method (inherited from SAConnectionStringStringBuilderBase)</td>
<td>Removes the entry with the specified key from the SAConnectionStringStringBuilder instance.</td>
</tr>
<tr>
<td>SetUseLongNameAsKeyword method (inherited from SAConnectionStringStringBuilderBase)</td>
<td>Sets a boolean value that indicates whether long connection parameter names are used in the connection string. Long connection parameter names are used by default.</td>
</tr>
<tr>
<td>ShouldSerialize method (inherited from SAConnectionStringStringBuilderBase)</td>
<td>Indicates whether the specified key exists in this SAConnectionStringStringBuilder instance.</td>
</tr>
</tbody>
</table>
**ToString method**
Converts the TcpOptionsBuilder object to a string representation.

**TryGetValue method** (inherited from SAConnectionStringBuilder)
Retrieves a value corresponding to the supplied key from this SA-ConnectionStringBuilder.

---

See also
- “SATcpOptionsBuilder class” on page 429
- “SAConnection class” on page 246

---

**SATcpOptionsBuilder constructors**

Initializes a new instance of the “SATcpOptionsBuilder class” on page 429.

**SATcpOptionsBuilder() constructor**

Initializes an SATcpOptionsBuilder object.

**Syntax**

*Visual Basic*

Public Sub New()

*C#*

public SATcpOptionsBuilder();

**Remarks**

**Restrictions:** 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();

**See also**
- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429
- “SATcpOptionsBuilder constructors” on page 432
**SATcpOptionsBuilder(String) constructor**

Initializes an SATcpOptionsBuilder object.

**Syntax**

**Visual Basic**

```vbnet
Public Sub New(_
    ByVal options As String _
)
```

**C#**

```csharp
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**

**Restrictions:** The SATcpOptionsBuilder class is not available in the .NET Compact Framework 2.0.

**Example**

The following statement initializes an SATcpOptionsBuilder object.

```csharp
SATcpOptionsBuilder options = new SATcpOptionsBuilder( );
```

**See also**

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429
- “SATcpOptionsBuilder constructors” on page 432

**Broadcast property**

Gets or sets the Broadcast option.

**Syntax**

**Visual Basic**

```vbnet
Public Property Broadcast As String
```

**C#**

```csharp
public string Broadcast { get; set; }
```
See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**BroadcastListener property**

Gets or sets the BroadcastListener option.

**Syntax**

Visual Basic

```vbnet
Public Property BroadcastListener As String
```

C#

```csharp
public string BroadcastListener { get; set; }
```

See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**ClientPort property**

Gets or sets the ClientPort option.

**Syntax**

Visual Basic

```vbnet
Public Property ClientPort As String
```

C#

```csharp
public string ClientPort { get; set; }
```

See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**DoBroadcast property**

Gets or sets the DoBroadcast option.
Syntax
   Visual Basic
   Public Property DoBroadcast As String

   C#
   public string DoBroadcast { get; set; }

See also
   ● “SATcpOptionsBuilder class” on page 429
   ● “SATcpOptionsBuilder members” on page 429

Host property

   Gets or sets the Host option.

Syntax
   Visual Basic
   Public Property Host As String

   C#
   public string Host { get; set; }

See also
   ● “SATcpOptionsBuilder class” on page 429
   ● “SATcpOptionsBuilder members” on page 429

IPV6 property

   Gets or sets the IPV6 option.

Syntax
   Visual Basic
   Public Property IPV6 As String

   C#
   public string IPV6 { get; set; }

See also
   ● “SATcpOptionsBuilder class” on page 429
   ● “SATcpOptionsBuilder members” on page 429
**LDAP property**

Gets or sets the LDAP option.

**Syntax**

**Visual Basic**

Public Property LDAP As String

**C#**

public string LDAP { get; set; }

**See also**

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**LocalOnly property**

Gets or sets the LocalOnly option.

**Syntax**

**Visual Basic**

Public Property LocalOnly As String

**C#**

public string LocalOnly { get; set; }

**See also**

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**MyIP property**

Gets or sets the MyIP option.

**Syntax**

**Visual Basic**

Public Property MyIP As String

**C#**

public string MyIP { get; set; }
ReceiveBufferSize property

Gets or sets the ReceiveBufferSize option.

Syntax
Visual Basic
Public Property ReceiveBufferSize As Integer

C#
public int ReceiveBufferSize { get; set; }

See also
● “SATcpOptionsBuilder class” on page 429
● “SATcpOptionsBuilder members” on page 429

SendBufferSize property

Gets or sets the Send BufferSize option.

Syntax
Visual Basic
Public Property SendBufferSize As Integer

C#
public int SendBufferSize { get; set; }

See also
● “SATcpOptionsBuilder class” on page 429
● “SATcpOptionsBuilder members” on page 429

ServerPort property

Gets or sets the ServerPort option.
Syntax

**Visual Basic**

Public Property **ServerPort** As String

**C#**

public string **ServerPort** { get; set; }

See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**TDS property**

Gets or sets the TDS option.

Syntax

**Visual Basic**

Public Property **TDS** As String

**C#**

public string **TDS** { get; set; }

See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429

**Timeout property**

Gets or sets the Timeout option.

Syntax

**Visual Basic**

Public Property **Timeout** As Integer

**C#**

public int **Timeout** { get; set; }

See also

- “SATcpOptionsBuilder class” on page 429
- “SATcpOptionsBuilder members” on page 429
VerifyServerName property

Gets or sets the VerifyServerName option.

Syntax

Visual Basic

Public Property VerifyServerName As String

C#

public string VerifyServerName { get; set; }

See also

● “SATcpOptionsBuilder class” on page 429
● “SATcpOptionsBuilder members” on page 429

ToString method

Converts the TcpOptionsBuilder object to a string representation.

Syntax

Visual Basic

Public Overrides Function ToString() As String

C#

public override string ToString();

Return value

The options string being built.

See also

● “SATcpOptionsBuilder class” on page 429
● “SATcpOptionsBuilder members” on page 429

SATransaction class

Represents a SQL transaction. This class cannot be inherited.

Syntax

Visual Basic

Public NotInheritable Class SATransaction
Inherits DbTransaction
public sealed class SATransaction : DbTransaction

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 136 and “Inserting, updating, and deleting rows
using the SACommand object” on page 120.

See also

● “SATransaction members” on page 440
● “BeginTransaction() method” on page 254
● “BeginTransaction(SAIsolationLevel) method” on page 256
● “Transaction property” on page 214

SATransaction members

Public properties

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection property</td>
<td>The SAConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.</td>
</tr>
<tr>
<td>IsolationLevel property</td>
<td>Specifies the isolation level for this transaction.</td>
</tr>
<tr>
<td>SAIIsolationLevel property</td>
<td>Specifies the isolation level for this transaction.</td>
</tr>
</tbody>
</table>

Public methods

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit method</td>
<td>Commits the database transaction.</td>
</tr>
<tr>
<td>Dispose (inherited from DbTransaction)</td>
<td>Releases the unmanaged resources used by the DbTransaction.</td>
</tr>
<tr>
<td>Rollback methods</td>
<td>Rolls back a transaction from a pending state.</td>
</tr>
<tr>
<td>Save method</td>
<td>Creates a savepoint in the transaction that can be used to roll back a portion of the transaction, and specifies the savepoint name.</td>
</tr>
</tbody>
</table>
Connection property

The SAConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax

Visual Basic

Public Readonly Property Connection As SAConnection

C#

public 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.

See also

● “SATransaction class” on page 439
● “BeginTransaction() method” on page 254
● “BeginTransaction(SAIsolationLevel) method” on page 256
● “Transaction property” on page 214

IsolationLevel property

Specifies the isolation level for this transaction.

Syntax

Visual Basic

Public Overrides Readonly Property IsolationLevel As IsolationLevel

C#

public override IsolationLevel IsolationLevel { get; }
Property value

The isolation level for this transaction. This can be one of:

- ReadCommitted
- ReadUncommitted
- RepeatableRead
- Serializable
- Snapshot
- ReadOnlySnapshot
- StatementSnapshot

The default is ReadCommitted.

See also

- “SATransaction class” on page 439
- “SATransaction members” on page 440

SAIsolationLevel property

Specifies the isolation level for this transaction.

Syntax

Visual Basic

Public Readonly Property SAIsolationLevel As SAIsolationLevel

C#

public SAIsolationLevel SAIsolationLevel { get;}

Property value

The IsolationLevel for this transaction. This can be one of:

- Chaos
- Read ReadCommitted
- ReadOnlySnapshot
- ReadUncommitted
- RepeatableRead
- Serializable
- Snapshot
- StatementSnapshot
- Unspecified

The default is ReadCommitted.

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.
Commit method

Commits the database transaction.

Syntax

Visual Basic
Public Overrides Sub Commit()

C#
public override void Commit();

Rollback methods

Rolls back a transaction from a pending state.

Rollback() method

Rolls back a transaction from a pending state.

Syntax

Visual Basic
Public Overrides Sub Rollback()

C#
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.

Syntax

Visual Basic

Public Sub Rollback(
    ByVal savePoint As String 
)

C#

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).

See also

- “SATransaction class” on page 439
- “SATransaction members” on page 440
- “Rollback methods” on page 443

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.

Syntax

Visual Basic

Public Sub Save(
    ByVal savePoint As String 
)

C#

```csharp
public void Save(
    string savePoint
);
```

**Parameters**
- **savePoint**  The name of the savepoint to which to roll back.

**See also**
- “SATransaction class” on page 439
- “SATransaction members” on page 440
SQL Anywhere OLE DB and ADO development

Contents

Introduction to OLE DB ................................................................. 448
ADO programming with SQL Anywhere ........................................ 449
Setting up a Microsoft Linked Server using OLE DB ...................... 456
Supported OLE DB interfaces .......................................................... 457
**Introduction to 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 OLE DB 2.5 and later.

For a list of supported platforms, see [http://www.sybase.com/detail?id=1061806](http://www.sybase.com/detail?id=1061806).

**Distributed transactions**

The OLE DB driver can be used as a resource manager in a distributed transaction environment.

For more information, see “Three-tier computing and distributed transactions” on page 65.
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 `samples-dir\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 Command1 on a form, and pasting the routine into its Click event. Run the program and click the button to connect and then disconnect.

```vbnet
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 11 Demo"
myConn.Open()
MsgBox("Connection succeeded")
myConn.Close()
Exit Sub

HandleError:
   MsgBox(Err.ToString(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.

When the SAOLEDB provider is installed, it 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.

**To 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 dboledb11.dll
regsvr32 dboledba11.dll
```

For more information about connecting to a database using OLE DB, see “Connecting to a database using OLE DB” [SQL Anywhere Server - Database Administration].

**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 Command2 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.

```vbnet
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 11 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.

For information about using transactions within ADO, see “Using transactions” on page 454.

You can also perform updates through a cursor.

For more information, see “Updating data through a cursor” on page 453.

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 11 Demo"
  myConn.CursorLocation = _
    ADODB.CursorLocationEnum.adUseServer
  myConn.Mode = _

  "ADO programming with SQL Anywhere"
ADODB.ConnectModeEnum.adModeReadWrite
myConn.IsolationLevel =
    ADODB.IsolationLevelEnum.adXactCursorStability
myConn.Open()

'Execute a query
myRS = New ADODB.Recordset
myRS.CacheSize = 50
myRS.let_Source("SELECT * FROM Customers")
myRS.let_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 “Working with the Recordset
object” on page 452.

Working with 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 39.
The available cursor types, the corresponding cursor type constants, and the SQL Anywhere types they are
equivalent to, are as follows:
### ADO cursor type

<table>
<thead>
<tr>
<th>ADO cursor type</th>
<th>ADO constant</th>
<th>SQL Anywhere type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic cursor</td>
<td>adOpenDynamic</td>
<td>Dynamic scroll cursor</td>
</tr>
<tr>
<td>Keyset cursor</td>
<td>adOpenKeyset</td>
<td>Scroll cursor</td>
</tr>
<tr>
<td>Static cursor</td>
<td>adOpenStatic</td>
<td>Insensitive cursor</td>
</tr>
<tr>
<td>Forward only</td>
<td>adOpenForwardOnly</td>
<td>No-scroll cursor</td>
</tr>
</tbody>
</table>

For information about choosing a cursor type that is suitable for your application, see “Choosing cursor types” on page 39.

### Sample code

The following code sets the cursor type for an ADO Recordset object:

```vba
Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

### 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.

```vba
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 11 Demo"
    myConn.Open()
    myConn.BeginTrans()
    SQLString = "SELECT * FROM Customers"
    myRS.Open(SQLString, myConn, _
        ADODB.CursorTypeEnum.adOpenDynamic, _
        ADODB.LockTypeEnum.adLockBatchOptimistic)

    If myRS.BOF And myRS.EOF Then
        MsgBox("Recordset is empty!", 16, "Empty Recordset")
    Else
```

ADO programming with SQL Anywhere 11.0.1 Copyright © 2009, iAnywhere Solutions, Inc.
```vbscript
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.

Using 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:

<table>
<thead>
<tr>
<th>ADO isolation level</th>
<th>Constant</th>
<th>SQL Anywhere level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unspecified</td>
<td>adXactUnspecified</td>
<td>Not applicable. Set to 0</td>
</tr>
<tr>
<td>Chaos</td>
<td>adXactChaos</td>
<td>Unsupported. Set to 0</td>
</tr>
<tr>
<td>Browse</td>
<td>adXactBrowse</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Using 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:

<table>
<thead>
<tr>
<th>ADO isolation level</th>
<th>Constant</th>
<th>SQL Anywhere level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unspecified</td>
<td>adXactUnspecified</td>
<td>Not applicable. Set to 0</td>
</tr>
<tr>
<td>Chaos</td>
<td>adXactChaos</td>
<td>Unsupported. Set to 0</td>
</tr>
<tr>
<td>Browse</td>
<td>adXactBrowse</td>
<td>0</td>
</tr>
<tr>
<td>ADO isolation level</td>
<td>Constant</td>
<td>SQL Anywhere level</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Read uncommitted</td>
<td>adXactReadUncommitted</td>
<td>0</td>
</tr>
<tr>
<td>Cursor stability</td>
<td>adXactCursorStability</td>
<td>1</td>
</tr>
<tr>
<td>Read committed</td>
<td>adXactReadCommitted</td>
<td>1</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>adXactRepeatableRead</td>
<td>2</td>
</tr>
<tr>
<td>Isolated</td>
<td>adXactIsolated</td>
<td>3</td>
</tr>
<tr>
<td>Serializable</td>
<td>adXactSerializable</td>
<td>3</td>
</tr>
<tr>
<td>Snapshot</td>
<td>2097152</td>
<td>4</td>
</tr>
<tr>
<td>Statement snapshot</td>
<td>4194304</td>
<td>5</td>
</tr>
<tr>
<td>Readonly statement snapshot</td>
<td>8388608</td>
<td>6</td>
</tr>
</tbody>
</table>

For more information about isolation levels, see “Isolation levels and consistency” [SQL Anywhere Server - SQL Usage].
Setting up 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 4-part table referencing syntax or the Microsoft OPENQUERY SQL function. An example of the 4-part syntax follows.

```
SELECT * FROM SADATABASE..GROUPO.Customers
```

In this example, SADATABASE is the name of the Linked Server, GROUPO is the table owner in the SQL Anywhere database, and Customers is the table name in the SQL Anywhere database. The catalog name is omitted (as indicated by two consecutive dots) since catalog names are not a feature of SQL Anywhere databases.

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.

To set up a Linked Server that uses the SQL Anywhere OLE DB provider, a few steps must be followed.

**To set up a Linked Server**

1. Fill in the **General** page.
   
   The **Linked Server** field on the **General** page should contain a **Linked Server** name (like SADATABASE used above). The **Other Data Source** option should be chosen, and SQL Anywhere OLE DB Provider should be chosen from the list. The **Product Name** field should contain an ODBC data source name (for example, SQL Anywhere 11 Demo). The **Provider String** field can contain additional connection parameters such as user ID and password (for example, uid=DBA;pwd=sql). Other fields, such as **Data Source**, on the **General** page should be left empty.

2. 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. In SQL Server 2005, there is a global **Allow Inprocess** checkbox when you right-click the SAOLEDB provider in the **Linked Servers/Providers** tree view and choose **Properties**. If the **InProcess** option is not chosen, queries fail.

3. Choose 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. These check boxes are found on the **Server Options** page. In SQL Server 2005, 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.
## 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.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Purpose</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAccessor</td>
<td>Define bindings between client memory and data store values.</td>
<td>DBACCESSOR_PASSBYREF not supported. DBACCESSOR_OPTIMIZED not supported.</td>
</tr>
<tr>
<td>IAlterIndex</td>
<td>Alter tables, indexes, and columns.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IAlterTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IChapteredRowset</td>
<td>A chaptered rowset allows rows of a rowset to be accessed in separate chapters.</td>
<td>Not supported. SQL Anywhere does not support chaptered rowsets.</td>
</tr>
<tr>
<td>IColumnsInfo</td>
<td>Get simple information about the columns in a rowset.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IColumnsRowset</td>
<td>Get information about optional metadata columns in a rowset, and get a rowset of column metadata.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ICommand</td>
<td>Execute SQL statements.</td>
<td>Does not support calling. ICommandProperties: GetProperties with DBPROP- SET_PROPERTIESINERROR to find properties that could not have been set.</td>
</tr>
<tr>
<td>ICommandPersist</td>
<td>Persist the state of a command object (but not any active rowsets). These persistent command objects can subsequently be enumerated using the PRO- CEDURES or VIEWS rowset.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ICommandPrepare</td>
<td>Prepare commands.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ICommandProperties</td>
<td>Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.</td>
<td>Supported.</td>
</tr>
<tr>
<td>Interface</td>
<td>Purpose</td>
<td>Limitations</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ICommandText</td>
<td>Set the SQL statement text for ICommand.</td>
<td>Only the DBGUID_DEFAULT SQL dialect is supported.</td>
</tr>
<tr>
<td>ICommandWithParameters</td>
<td>Set or get parameter information for a command.</td>
<td>No support for parameters stored as vectors of scalar values. No support for BLOB parameters.</td>
</tr>
<tr>
<td>IConvertType</td>
<td></td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBAsynchNotify</td>
<td>Asynchronous processing.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IDBAasyncStatus</td>
<td>Notify client of events in the asynchronous processing of data source initialization, populating rowsets, and so on.</td>
<td></td>
</tr>
<tr>
<td>IDBCreateCommand</td>
<td>Create commands from a session.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBCreateSession</td>
<td>Create a session from a data source object.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBDataSourceAdmin</td>
<td>Create/destroy/modify data source objects, which are COM objects used by clients. This interface is not used to manage data stores (databases).</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IDBInfo</td>
<td>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.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBInitialize</td>
<td>Initialize data source objects and enumerators.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBProperties</td>
<td>Manage properties on a data source object or enumerator.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IDBSchemaRowset</td>
<td>Get information about system tables, in a standard form (a rowset).</td>
<td>Supported.</td>
</tr>
<tr>
<td>Interface</td>
<td>Purpose</td>
<td>Limitations</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>IErrorInfo</td>
<td>ActiveX error object support.</td>
<td>Supported</td>
</tr>
<tr>
<td>IErrorLookup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IErrorRecords</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IGetDataSource</td>
<td>Returns an interface pointer to the session’s data source object.</td>
<td>Supported</td>
</tr>
<tr>
<td>IIndexDefinition</td>
<td>Create or drop indexes in the data store.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IMultipleResults</td>
<td>Retrieve multiple results (rowsets or row counts) from a command.</td>
<td>Supported</td>
</tr>
<tr>
<td>IOpenRowset</td>
<td>Non-SQL way to access a database table by its name.</td>
<td>Supported. Opening a table by its name is supported, not by a GUID.</td>
</tr>
<tr>
<td>IRowset</td>
<td>Access rowsets.</td>
<td>Supported</td>
</tr>
<tr>
<td>IRowsetChange</td>
<td>Allow changes to rowset data, reflected back to the data store.</td>
<td>Supported</td>
</tr>
<tr>
<td>IRowsetChapterMember</td>
<td>Access chaptered/hierarchical rowsets.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetCurrentIndex</td>
<td>Dynamically change the index for a rowset.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetFind</td>
<td>Find a row within a rowset matching a specified value.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetIdentity</td>
<td>Compare row handles.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetIndex</td>
<td>Access database indexes.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetInfo</td>
<td>Find information about rowset properties or to find the object that created the rowset.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IRowsetLocate</td>
<td>Position on rows of a rowset, using bookmarks.</td>
<td>Supported.</td>
</tr>
<tr>
<td>Interface</td>
<td>Purpose</td>
<td>Limitations</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>IRowsetNotify</td>
<td>Provides a COM callback interface for rowset events.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IRowsetRefresh</td>
<td>Get the latest value of data that is visible to a transaction.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetResynch</td>
<td>Old OLEDB 1.x interface, superseded by IRowsetRefresh.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetScroll</td>
<td>Scroll through rowset to fetch row data.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IRowsetUpdate</td>
<td>Delay changes to rowset data until Update is called.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IRowsetView</td>
<td>Use views on an existing rowset.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>ISequentialStream</td>
<td>Retrieve a BLOB column.</td>
<td>Supported for reading only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No support for SetData with this interface.</td>
</tr>
<tr>
<td>ISessionProperties</td>
<td>Get session property information.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ISourcesRowset</td>
<td>Get a rowset of data source objects and enumerators.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ISQLErrorInfo</td>
<td>ActiveX error object support.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ISupportErrorInfo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITableDefinition</td>
<td>Create, drop, and alter tables, with constraints.</td>
<td>Supported.</td>
</tr>
<tr>
<td>ITableDefinitionWithConstraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITransaction</td>
<td>Commit or abort transactions.</td>
<td>Not all the flags are supported.</td>
</tr>
<tr>
<td>ITransactionJoin</td>
<td>Support distributed transactions.</td>
<td>Not all the flags are supported.</td>
</tr>
<tr>
<td>ITransactionLocal</td>
<td>Handle transactions on a session.</td>
<td>Supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not all the flags are supported.</td>
</tr>
<tr>
<td>ITransactionOptions</td>
<td>Get or set options on a transaction.</td>
<td>Supported.</td>
</tr>
<tr>
<td>IViewChapter</td>
<td>Work with views on an existing rowset, specifically to apply post-processing filters/sorting on rows.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>Interface</td>
<td>Purpose</td>
<td>Limitations</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>IViewFilter</td>
<td>Restrict contents of a rowset to rows matching a set of conditions.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IViewRowset</td>
<td>Restrict contents of a rowset to rows matching a set of conditions, when opening a rowset.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>IViewSort</td>
<td>Apply sort order to a view.</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>

Supported OLE DB interfaces
SQL Anywhere ODBC API

Contents

Introduction to ODBC ................................................................................................ 464
Building ODBC applications ...................................................................................... 466
ODBC samples .......................................................................................................... 471
ODBC handles ........................................................................................................... 473
Choosing an ODBC connection function ................................................................... 476
SQL Anywhere connection attributes ........................................................................ 479
Executing SQL statements ........................................................................................ 481
64-bit ODBC considerations ...................................................................................... 485
Data alignment requirements .................................................................................... 489
Working with result sets ............................................................................................. 491
Calling stored procedures .......................................................................................... 496
Handling errors .......................................................................................................... 498
Introduction to ODBC

The Open Database Connectivity (ODBC) interface is an application programming interface defined by Microsoft Corporation as a standard interface to database management systems on Windows operating systems. ODBC is a call-based interface.

To write ODBC applications for SQL Anywhere, you need:

- SQL Anywhere.
- A 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.

Supported platforms

SQL Anywhere supports the ODBC API on Unix and Windows Mobile, in addition to Windows. Having multi-platform ODBC support makes portable database application development much easier.

For information about enlisting the ODBC driver in distributed transactions, see “Three-tier computing and distributed transactions” on page 65.

See also

- Microsoft Open Database Connectivity (ODBC)

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.

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.

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.

**ODBC backward compatibility**
Applications developed using older versions of ODBC continue to work with SQL Anywhere and the newer ODBC driver manager. The new ODBC features are not provided for older applications.

**The ODBC driver manager**
Microsoft Windows includes an ODBC driver manager. For Unix, an ODBC driver manager is supplied with SQL Anywhere.
Building ODBC applications

This section describes how to compile and link simple ODBC applications.

Including the ODBC header file

Every 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.

To include the ODBC header file in a 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:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Include line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td><code>#include &quot;ntodbc.h&quot;</code></td>
</tr>
<tr>
<td>Unix</td>
<td><code>#include &quot;unixodbc.h&quot;</code></td>
</tr>
<tr>
<td>Windows Mobile</td>
<td><code>#include &quot;ntodbc.h&quot;</code></td>
</tr>
</tbody>
</table>

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 of our supported platforms
   - 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.

For Windows Mobile information, see “Linking ODBC applications on Windows Mobile” on page 467.

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 `dbodbc11.dll`. 
To 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:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Import library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows (32-bit)</td>
<td>Lib\odbc32.lib</td>
</tr>
<tr>
<td>Windows (64-bit)</td>
<td>Lib\x64\odbc32.lib</td>
</tr>
</tbody>
</table>

Here is an example for a command prompt.

```bash
set LIB=%LIB%;C:\mssdk\v6.1\lib
```

2. To compile and link a simple ODBC application using the Microsoft compile and link tool, you can issue a single command from the command prompt. The following example compiles and links the application stored in odbc.c.

```bash
cl odbc.c /Ic:\sa11\SDK\Include odbc32.lib
```

Linking ODBC applications on Windows Mobile

On Windows Mobile operating systems, there is no ODBC driver manager. The import library (dbodbc11.lib) defines entry points directly into the SQL Anywhere ODBC driver dbodbc11.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 for the DRIVER= parameter in the connection string supplied to the SQLDriverConnect function. The following is an example.

```bash
szConnStrIn = "driver=opath\dbodbc11.dll;dbf=\samples-dir\demo.db"
```

Here, opath is the full path to the Windows directory on the Windows Mobile device. For example:

\Windows

To 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 in http://www.sybase.com/detail?id=1002288.

The sample program (odbc_sample.cpp) uses a File Data Source (FileDSN connection parameter) called SQL Anywhere 11 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.
For information about the default location of `samples-dir`, see “Samples directory” [SQL Anywhere Server - Database Administration].

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 `samples-dir\SQLAnywhere\C\odbc.c` illustrates how to use the Unicode ODBC features.

Linking ODBC applications on Unix

An ODBC driver manager 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 multi-threaded 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 multi-threaded library.

The ODBC drivers are the following files:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Threading model</th>
<th>ODBC driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>(all Unix except Mac OS X and HP-UX)</td>
<td>Generic</td>
<td><code>libdbodbc11.so (libdbodbc11.so.1)</code></td>
</tr>
<tr>
<td>(all Unix except Mac OS X and HP-UX)</td>
<td>Single threaded</td>
<td><code>libdbodbc11_n.so (libdbodbc11_n.so.1)</code></td>
</tr>
<tr>
<td>(all Unix except Mac OS X and HP-UX)</td>
<td>Multi-threaded</td>
<td><code>libdbodbc11_r.so (libdbodbc11_r.so.1)</code></td>
</tr>
<tr>
<td>HP-UX</td>
<td>Generic</td>
<td><code>libdbodbc11.sl (libdbodbc11.sl.1)</code></td>
</tr>
<tr>
<td>HP-UX</td>
<td>Single threaded</td>
<td><code>libdbodbc11_n.sl (libdbodbc11_n.sl.1)</code></td>
</tr>
<tr>
<td>HP-UX</td>
<td>Multi-threaded</td>
<td><code>libdbodbc11_r.sl (libdbodbc11_r.sl.1)</code></td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Generic</td>
<td><code>libdbodbc11.dylib</code></td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Single threaded</td>
<td><code>libdbodbc11_n.dylib</code></td>
</tr>
</tbody>
</table>
Operating system | Threading model | ODBC driver
--- | --- | ---
Mac OS X | Multi-threaded | libdbodbc11_r.dylib

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:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Threading model</th>
<th>ODBC driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X</td>
<td>Single threaded</td>
<td>dbodbc11.bundle</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Multi-threaded</td>
<td>dbodbc11_r.bundle</td>
</tr>
</tbody>
</table>

**To link an ODBC application (Unix)**

1. Link your application against the generic ODBC driver libdbodbc11.
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 “Using ODBC data sources on Unix” [SQL Anywhere Server - Database Administration].

**Using the SQL Anywhere ODBC driver manager on Unix**

SQL Anywhere includes an ODBC driver manager for Unix. The libdbodm11 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.
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.

**Using the unixODBC driver manager**

Versions of the unixODBC release prior to version 2.2.13 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 2.2.13, when it is released, will 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. See “64-bit ODBC considerations” on page 485.

3. You must not use versions of the unixODBC driver manager prior to release 2.2.13. Instead, you can link directly to the SQL Anywhere ODBC driver. For example, ensure that the libodbc shared object is linked to the SQL Anywhere driver.

   \[
   \text{libodbc.so.1} \rightarrow \text{libdbodbc11_r.so.1}
   \]

   Alternatively, you can use the SQL Anywhere driver manager on platforms where it is available. See “Using the SQL Anywhere ODBC driver manager on Unix” on page 469.

For more information, see “Linking ODBC applications on Unix” on page 468.
ODBC samples

Several ODBC samples are included with SQL Anywhere. You can find the samples in the samples-dir \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 samples-dir \SQLAnywhere\C\odbc.c. This program performs the same actions as the embedded SQL dynamic cursor example program that is in the same directory.

For a description of the associated embedded SQL program, see “Sample embedded SQL programs” on page 538.

Building the sample ODBC program

The ODBC sample program is located in samples-dir\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.

To build the sample ODBC program

1. Open a command prompt and change directory to the samples-dir\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\v6.1
   build64

To build the sample ODBC program for Unix

1. Open a command shell and change directory to the samples-dir\SQLAnywhere\C directory.
2. Run the build.sh shell script.

Running the sample ODBC program

To run the ODBC sample

1. Start the program:
   
   ● For 32-bit Windows, run the file samples-dir\SQLAnywhere\C\odbcwin.exe.
   
   ● For 64-bit Windows, run the file samples-dir\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:

  ```c
  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:

  ```c
  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:

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

### Allocating ODBC handles

The handle types required for ODBC programs are as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Handle type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>SQLHENV</td>
</tr>
<tr>
<td>Connection</td>
<td>SQLHDBC</td>
</tr>
<tr>
<td>Statement</td>
<td>SQLHSTMT</td>
</tr>
</tbody>
</table>
**To 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 a full description, see `SQLAllocHandle` in the Microsoft *ODBC Programmer's Reference*.

2. Use the handle in subsequent function calls.

3. Free the object using SQLFreeHandle.

   SQLFreeHandle takes the following parameters:
   
   - an identifier for the type of item being freed
   - the handle of the item being freed
   
   For a full description, see `SQLFreeHandle` in the Microsoft ODBC Programmer's Reference.

**Example**

The following code fragment allocates and frees an environment handle:

```c
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 );
```

For more information about return codes and error handling, see “Handling errors” on page 498.

**A first ODBC example**

The following is a simple ODBC program that connects to the SQL Anywhere sample database and immediately disconnects.

You can find this sample in `samples-dir\SQLAnywhere\ODBCConnect\odbcconnect.cpp`.

```c
#include <stdio.h>
#include "ntodbc.h"
```
```c
int main(int argc, char* argv[]) {
    SQLHENV env;
    SQLHDBC dbc;
    SQLRETURN retcode;

    retcode = SQLAllocHandle( SQL_HANDLE_ENV,
                              SQL_NULL_HANDLE,
                              &env );
    if (retcode == SQL_SUCCESS
        || retcode == SQL_SUCCESS_WITH_INFO) {
        printf( "env allocated\n" );
        /* Set the ODBC version environment attribute */
        retcode = SQLSetEnvAttr( env,
                                 SQL_ATTR_ODBC_VERSION,
                                 (void*)SQL_OV_ODBC3, 0);
        retcode = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
        if (retcode == SQL_SUCCESS
            || retcode == SQL_SUCCESS_WITH_INFO) {
            printf( "dbc allocated\n" );
            retcode = SQLConnect( dbc,
                                 (SQLCHAR*) "SQL Anywhere 11 Demo", SQL_NTS,
                                 (SQLCHAR*) "DBA", SQL_NTS,
                                 (SQLCHAR*) "sql", SQL_NTS );
            if (retcode == SQL_SUCCESS
                || retcode == SQL_SUCCESS_WITH_INFO) {
                printf( "Successfully connected\n" );
            } else {
                SQLDisconnect( dbc );
            }
            SQLFreeHandle( SQL_HANDLE_DBC, dbc );
        } else {
            SQLDisconnect( dbc );
        }
    }
    SQLFreeHandle( SQL_HANDLE_ENV, env );
    return 0;
}
```
Choosing an ODBC connection function

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 Programmer's Reference.

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

- **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 Programmer's Reference.

For a complete list of connection parameters that can be used in connection strings, see “Connection parameters” [SQL Anywhere Server - Database Administration].

Establishing a connection

Your application must establish a connection before it can perform any database operations.

To 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_ODBC, 0);

3. 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.

4. Allocate an ODBC connection item.

   For example:

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

5. 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 );

   For more information, see “Setting connection attributes” on page 477.

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 11 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 samples-dir\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.

   For more information, see “Setting connection attributes” on page 477.

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

Many aspects of the connection can be controlled through the connection parameters. For information, see “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 Programmer's Reference.

Threads and connections in ODBC applications

You can develop multi-threaded 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.
The SQL Anywhere ODBC driver supports some extended connection attributes.

- **SA_REGISTER_MESSAGE_CALLBACK** Messages can be sent to the client application from the server using the SQL MESSAGE statement. A message handler routine can created to intercept these messages. The message handler callback prototype is as follows:

  ```c
  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.

  A SQLCODE associated with the message may be provided in `code`. 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.

  ```c
  memcpy( mybuff, msg, len );
  mybuff[len] = '\0';
  ```

  To register the message handler in ODBC, call the `SQLSetConnectAttr` function as follows:

  ```c
  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:

  ```c
  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.

  ```c
  SQLHDBC callback_hdbc = NULL;
  rc = SQLGetConnectAttr( 
      hdbc,
  ```
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:

  ```c
  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 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.

- **SA_SQL_ATTR_TXN_ISOLATION**  
  This is used to set an extended transaction isolation level. The following example sets a snapshot isolation level:

  ```c
  SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
  SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION, 
                    SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
  ```

  For more information, see “Choosing ODBC transaction isolation level” on page 491.
Executing 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.
  
  For more information, see “Executing prepared statements” on page 483.

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 482.

To 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`:
   
   ```c
   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:
   
   ```c
   SQLCHAR deletestmt[ STMT_LEN ] =
   "DELETE FROM Departments WHERE DepartmentID = 201";
   SQLExecDirect( stmt, deletestmt, SQL_NTS ) ;
   ```

   For a complete sample with error checking, see `samples-dir\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.

   For more information about SQLExecDirect, see `SQLExecDirect` in the Microsoft *ODBC Programmer's Reference*. 
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.

To 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 
   "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 );
   ```

   Bind parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once. For more information, see “Executing prepared statements” on page 483.
The above code fragments do not include error checking. For a complete sample, including error checking, see `samples-dir\SQLAnywhere\ODBCExecute\odbcexecute.cpp`.

For more information about SQLExecDirect, see SQLExecDirect in the Microsoft ODBC Programmer's Reference.

## 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.

For an introduction to prepared statements, see “Preparing statements” on page 26.

### To execute a prepared SQL statement

1. Prepare the statement using SQLPrepare.

   For example, the following code fragment illustrates how to prepare an INSERT statement:

   ```c
   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.
   
   2. Set statement parameter values using SQLBindParameter.

   For example, the following function call sets the value of the DepartmentID variable:

   ```c
   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:

    ```c
    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 `samples-dir\SQLAnywhere\ODBCPrepare\odbcprepare.cpp`.

For more information about SQLPrepare, see **SQLPrepare** in the Microsoft *ODBC Programmer's Reference*. 
64-bit ODBC considerations

When you use an ODBC function like SQLBindCol, SQLBindParameter, or SQLGetData, some of the parameters are typed as SQLLEN or SQUULEN 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.

```c
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.

```c
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 ntdbc.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.

<table>
<thead>
<tr>
<th>ODBC API</th>
<th>64-bit platform</th>
<th>32-bit platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLINTEGER</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>SQLUINTEGER</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>SQLLEN</td>
<td>64 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>SQLULEN</td>
<td>64 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>SQLSETPOSROW</td>
<td>64 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1 485
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 since 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.

<table>
<thead>
<tr>
<th>ODBC API</th>
<th>Parameter (Documented Parameter Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLBindCol</td>
<td>SQLLEN BufferLength</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *Strlen_or_Ind</td>
</tr>
<tr>
<td>SQLBindParam</td>
<td>SQLULEN LengthPrecision</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *Strlen_or_Ind</td>
</tr>
<tr>
<td>SQLBindParameter</td>
<td>SQLULEN cbColDef (ColumnSize)</td>
</tr>
<tr>
<td></td>
<td>SQLLEN cbValueMax (BufferLength)</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *pcbValue (Strlen_or_IndPtr)</td>
</tr>
<tr>
<td>SQLColAttribute</td>
<td>SQLLEN *NumericAttribute</td>
</tr>
<tr>
<td>SQLColAttributes</td>
<td>SQLLEN *pfDesc</td>
</tr>
<tr>
<td>SQLDescribeCol</td>
<td>SQLULEN *ColumnSize (ColumnSizePtr)</td>
</tr>
<tr>
<td>SQLDescribeParam</td>
<td>SQLULEN *pcbParamDef (ParameterSizePtr)</td>
</tr>
<tr>
<td>SQLExtendedFetch</td>
<td>SQLLEN irow (FetchOffset)</td>
</tr>
<tr>
<td></td>
<td>SQLULEN *pcrow (RowCountPtr)</td>
</tr>
<tr>
<td>SQLFetchScroll</td>
<td>SQLLEN FetchOffset</td>
</tr>
<tr>
<td>SQLGetData</td>
<td>SQLLEN BufferLength</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)</td>
</tr>
<tr>
<td>SQLGetDescRec</td>
<td>SQLLEN *Length (LengthPtr)</td>
</tr>
<tr>
<td>SQLParamOptions</td>
<td>SQLULEN crow,</td>
</tr>
<tr>
<td></td>
<td>SQLULEN *pirow</td>
</tr>
<tr>
<td>ODBC API</td>
<td>Parameter (Documented Parameter Name)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>SQLPutData</td>
<td>SQLLEN Strlen_or_Ind</td>
</tr>
<tr>
<td>SQLRowCount</td>
<td>SQLLEN *RowCount (RowCountPtr)</td>
</tr>
<tr>
<td>SQLSetConnectOption</td>
<td>SQLULEN Value</td>
</tr>
<tr>
<td>SQLSetDescRec</td>
<td>SQLLEN Length</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *StringLength (StringLengthPtr)</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *Indicator (IndicatorPtr)</td>
</tr>
<tr>
<td>SQLSetParam</td>
<td>SQLULEN LengthPrecision</td>
</tr>
<tr>
<td></td>
<td>SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)</td>
</tr>
<tr>
<td>SQLSetPos</td>
<td>SQLSETPOSIROW irow (RowNumber)</td>
</tr>
<tr>
<td>SQLSetScrollOptions</td>
<td>SQLLEN crowKeyset</td>
</tr>
<tr>
<td>SQLSetStmtOption</td>
<td>SQLULEN Value</td>
</tr>
</tbody>
</table>

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/SQLUINTEGER. The same rule applies to the corresponding parameters for the SQLGetStmtAttr and SQLGetDescField functions.

<table>
<thead>
<tr>
<th>ODBC API</th>
<th>Type for Value/ValuePtr variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_FETCH_BOOKMARK_PTR)</td>
<td>SQLLEN * value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_MAX_LENGTH)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_PARAM_BIND_OFFSET_PTR)</td>
<td>SQLULEN * value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_PARAMS_PROCESSED_PTR)</td>
<td>SQLULEN * value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_PARAMSET_SIZE)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_ROW_ARRAY_SIZE)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_ROW_BIND_OFFSET_PTR)</td>
<td>SQLULEN * value</td>
</tr>
<tr>
<td>ODBC API</td>
<td>Type for Value/ValuePtr variable</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)</td>
<td>SQLULEN * value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_ARRAY_SIZE)</td>
<td>SQLULEN value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_BIND_OFFSET_PTR)</td>
<td>SQLLEN * value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_ROWS_PROCESSED_PTR)</td>
<td>SQLULEN * value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_DISPLAY_SIZE)</td>
<td>SQLLEN value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_INDICATOR_PTR)</td>
<td>SQLLEN * value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_LENGTH)</td>
<td>SQLLEN value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_OCTET_LENGTH)</td>
<td>SQLLEN value</td>
</tr>
<tr>
<td>SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)</td>
<td>SQLLEN * value</td>
</tr>
</tbody>
</table>

For more information, see the Microsoft article [ODBC 64-Bit API Changes in MDAC 2.7](#).
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 following table lists memory alignment requirements for processors such as Sun Sparc, Itanium-IA64, and ARM-based devices.

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Alignment required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_C_CHAR</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_BINARY</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_GUID</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_BIT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_STINYINT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_UTINYINT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_TINYINT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_DEFAULT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_C_SSHORT</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_USHORT</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_SHORT</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_DATE</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_TIME</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_TIMESTAMP</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_TYPE_DATE</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_TYPE_TIME</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_TYPE_TIMESTAMP</td>
<td>2</td>
</tr>
<tr>
<td>SQL_C_WCHAR</td>
<td>2 (buffer size must be a multiple of 2 on all platforms)</td>
</tr>
<tr>
<td>SQL_C_SLONG</td>
<td>4</td>
</tr>
<tr>
<td>C Data Type</td>
<td>Alignment required</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SQL_C_ULONG</td>
<td>4</td>
</tr>
<tr>
<td>SQL_C_LONG</td>
<td>4</td>
</tr>
<tr>
<td>SQL_C_FLOAT</td>
<td>4</td>
</tr>
<tr>
<td>SQL_C_DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>SQL_C_SBIGINT</td>
<td>8</td>
</tr>
<tr>
<td>SQL_C_UBIGINT</td>
<td>8</td>
</tr>
</tbody>
</table>

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.
ODBC applications use cursors to manipulate and update result sets. SQL Anywhere provides extensive support for different kinds of cursors and cursor operations.

For an introduction to cursors, see “Working with cursors” on page 32.

Choosing ODBC transaction isolation level

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 in cases where 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.

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

Example

The following fragment uses a snapshot isolation level:

```c
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION, SA_SQL_TXN_SNAPSHOT, SQL_IS_UINT );
```
Choosing 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 “Retrieving data” on page 493.

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 Programmer's Reference*.

Example

The following fragment requests a read-only, scrollable cursor:

```sql
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE, SQL_SCROLLABLE, SQL_IS_UINTEGER );
```
Retrieving data

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 `samples-dir\SQLAnywhere\ODBCSelect\odbcselect.cpp`.

```sql
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 11 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 modification statement on the columns in the result is meaningful, then positioned data modification statements can be carried out on the cursor.
  
  The ansi_update_constraints database option limits the type of queries that are updatable.
  
  For more information, see "ansi_update_constraints option [compatibility]" [SQL Anywhere Server - Database Administration].

- 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.

### Using 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.

For a full description of stored procedures and triggers, see “Using procedures, triggers, and batches” [SQL Anywhere Server - SQL Usage].

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

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.

```c
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

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.

```c
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()");
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 );
    }
}
```
Handling errors

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:

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_SUCCESS</td>
<td>No error.</td>
</tr>
<tr>
<td>SQL_SUCCESS_WITH_INFO</td>
<td>The function completed, but a call to SQLError will indicate a warning.</td>
</tr>
<tr>
<td></td>
<td>The most common case for this status is that a value being returned is too long for the buffer provided by the application.</td>
</tr>
<tr>
<td>SQL_ERROR</td>
<td>The function did not complete because of an error. Call SQLError to get more information about the problem.</td>
</tr>
<tr>
<td>SQL_INVALID_HANDLE</td>
<td>An invalid environment, connection, or statement handle was passed as a parameter.</td>
</tr>
<tr>
<td></td>
<td>This often happens if a handle is used after it has been freed, or if the handle is the null pointer.</td>
</tr>
<tr>
<td>SQL_NO_DATA_FOUND</td>
<td>There is no information available.</td>
</tr>
<tr>
<td></td>
<td>The most common use for this status is when fetching from a cursor; it indicates that there are no more rows in the cursor.</td>
</tr>
<tr>
<td>SQL_NEED_DATA</td>
<td>Data is needed for a parameter.</td>
</tr>
<tr>
<td></td>
<td>This is an advanced feature described in the ODBC SDK documentation under SQLParamData and SQLPutData.</td>
</tr>
</tbody>
</table>

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.
SQLerrorMessage 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:

```c
/* 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 ){
    SQLErrorMessage( 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 ) {
    SQLErrorMessage( 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:

```c
/* 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,
    &ernative, errmsg, sizeof(errmsg), &errmsglen);
/* Assume that print_error is defined */
print_error("Failed to delete items",
    errstate, ernative, errmsg);
return;
SQL Anywhere JDBC driver

Contents

Introduction to JDBC ................................................................. 502
Using the iAnywhere JDBC driver .............................................. 505
Using the jConnect JDBC driver ............................................... 507
Connecting from a JDBC client application .............................. 511
Using JDBC to access data ....................................................... 517
Using JDBC escape syntax ...................................................... 526
iAnywhere JDBC 3.0 API support ............................................ 529
Introduction to JDBC

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 supports and includes two JDBC drivers: the iAnywhere JDBC driver, which is a Type 2 JDBC driver, and 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 chapter in the directory `samples-dir\SQLAnywhere\JDBC`.

- **JDBC Specification**  You can find more on the JDBC Data Access API at Java SE Technologies - Database.

- **Required software**  You need TCP/IP to use the jConnect driver.

  The jConnect driver is available at jConnect for JDBC.

  For more information about the jConnect driver and its location, see “Using the jConnect JDBC driver” on page 507.

Choosing a JDBC driver

SQL Anywhere supports the following JDBC drivers:

- **iAnywhere JDBC 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 iAnywhere JDBC driver is the recommended JDBC driver for connecting to SQL Anywhere databases.

- **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 from jConnect for JDBC.

When choosing which driver to use, you should consider the following factors:

- **Features**  Both the iAnywhere JDBC driver and jConnect 6.0.5 are JDBC 3.0 compliant. However, the iAnywhere JDBC driver provides fully-scrollable cursors when connected to a SQL Anywhere database. The jConnect JDBC driver provides fully-scrollable cursors only when connected to an Adaptive Server Enterprise database.

  The JDBC 3.0 API documentation is available at JDBC Downloads. For a summary of the JDBC API methods supported by iAnywhere, see “iAnywhere JDBC 3.0 API support” on page 529.

- **Pure Java**  The jConnect driver is a pure Java solution. The iAnywhere JDBC driver requires the SQL Anywhere ODBC driver and is not a pure Java solution.

- **Performance**  The iAnywhere JDBC driver provides 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 iAnywhere JDBC driver and jConnect, see http://www.sybase.com/detail?id=1061806.

For information about using jConnect with Windows Mobile, see “Using 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 the iAnywhere 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".
Using the iAnywhere JDBC driver

The iAnywhere JDBC driver provides a JDBC driver that has some performance benefits and feature benefits compared to the pure Java jConnect JDBC driver, but which is not a pure-Java solution. The iAnywhere JDBC driver is recommended in most cases.

For information about choosing which JDBC driver to use, see “Choosing a JDBC driver” on page 502.

Loading the iAnywhere JDBC driver

Before you can use the iAnywhere JDBC driver in your application, you must load the appropriate driver. Load the iAnywhere JDBC 3.0 driver with the following statement:

```java
DriverManager.registerDriver((Driver)
    Class.forName("ianywhere.ml.jdbcodbc.jdbc3.IDriver").newInstance();
```

Using the newInstance method works around issues in some browsers.

- As the classes are loaded using Class.forName, the package containing the iAnywhere JDBC driver does not have to be imported using import statements.
- `jodbc.jar` must be in your classpath when you run the application.

```bash
set classpath=%classpath%;install-dir\java\jodbc.jar
```

Required files

The Java component of the iAnywhere JDBC driver is included in the `jodbc.jar` file installed into the `Java` subdirectory of your SQL Anywhere installation. For Windows, the native component is `dbjodbc11.dll` in the `bin32` or `bin64` subdirectory of your SQL Anywhere installation; for Unix, the native component is `libdbjodbc11.so`. This component must be in the system path. When deploying applications using this driver, you must also deploy the ODBC driver files.

Supplying a URL to the driver

To connect to a database via the iAnywhere JDBC driver, you need to supply a URL for the database. For example:

```java
Connection con = DriverManager.getConnection("jdbc:ianywhere:DSN=SQL Anywhere 11 Demo");
```

The URL contains `jdbc:ianywhere:` followed by a standard ODBC connection string. The connection string is commonly an ODBC data source, but you can also use explicit individual connection parameters, separated by semicolons, in addition to or instead of the data source. For more information about the parameters that you can use in a connection string, see “Connection parameters” [SQL Anywhere Server - Database Administration].
If you do not use a data source, you should specify the ODBC driver to use by including the DRIVER parameter in your connection string:

```java
Connection con = DriverManager.getConnection(
"jdbc:ianywhere:driver=SQL Anywhere 11;...");
```
Using the jConnect JDBC driver

SQL Anywhere supports one version of jConnect: jConnect 6.0.5. The jConnect driver is available as a separate download from jConnect for JDBC. 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

SQL Anywhere supports the following version of jConnect:

- **jConnect 6.0.5**  This version of jConnect is for developing JDK 1.4 or later applications. jConnect 6.0.5 is JDBC 3.0 compliant. jConnect 6.0.5 is supplied as a JAR file named jconn3.jar.

**Note**

For the purposes of this documentation, all the explanations and code samples provided assume that you are developing JDK 1.5 applications, and that you are using the jConnect 6.0.5 driver.

There is a copy of jconn3.jar in the install-dir\java folder. However, we recommend that you use the version of the file included with jConnect 6.0.5 since it will be current with the version of jConnect that you have installed.

Setting the class path for jConnect

For your application to use jConnect, the jConnect classes must be in your class 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 6.0.5 driver to an existing CLASSPATH environment variable (where path is your jConnect installation directory).

```sh
set classpath=path\jConnect-6_0\classes\jconn3.jar;%classpath%
```

Importing the jConnect classes

The classes in jConnect 6.0.5 are all in com.sybase.jdbc3.jdbc. You must import these classes at the beginning of each source file:

```java
import com.sybase.jdbc3.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.

### Windows Mobile
Do not add jConnect system objects to your Windows Mobile database. See “Using jConnect on Windows Mobile” [SQL Anywhere Server - Database Administration].

### 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. For information about backing up your database, see “Backup and data recovery” [SQL Anywhere Server - Database Administration].

### To add jConnect system objects to a database (Sybase Central)
1. Connect to the database from Sybase Central as a DBA user.
2. Select the database, if necessary, and then from the Tools menu choose SQL Anywhere 11 and then Upgrade Database.
3. Follow the instructions in the Upgrade Database Wizard.

### To add jConnect system objects to a database (dbupgrad)
- From a command prompt, run the following command:

  ```bash
  dbupgrad -c "connection-string"
  ```

  In this command, `connection-string` is a suitable connection string that enables access to a database and server as a DBA user.

### Loading the jConnect driver
Before you can use jConnect in your application, load the driver with the following statement:

```java
DriverManager.registerDriver( (Driver) 
  Class.forName("com.sybase.jdbc3.jdbc.SybDriver").newInstance()
);
```

Using the newInstance method works around issues in some browsers.

- As the classes are loaded using Class.forName, the package containing the jConnect driver does not have to be imported using import statements.
- To use jConnect 6.0.5, `jconn3.jar` must be in your classpath when you run the application. `jconn3.jar` is located in the `classes` subdirectory of your jConnect 6.0.5 installation (typically, `jConnect-6_0\classes`).
Supplying a URL to the driver

To connect to a database via jConnect, you need to supply a URL for the database. For example:

```java
Connection con = DriverManager.getConnection(
```

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.

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

```java
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:

```java
Connection con = DriverManager.getConnection(
```

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.

```java
import java.util.Properties;
```
DriverManager.registerDriver( DriverManager.registerDriver( (Driver)
    Class.forName( 
        "com.sybase.jdbc3.jdbc.SybDriver" ).newInstance() 
    );

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.

For more information about the utility database, see “Using the utility database” [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 Open Client and jConnect connections” [SQL Anywhere Server - Database Administration]
- “sp_tsql_environment system procedure” [SQL Anywhere Server - SQL Reference]
Connecting from a JDBC client application

Database metadata is always available when using the iAnywhere 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 “Using the jConnect JDBC driver” on page 507.

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 514.

Connection example code

The following is the source code for the methods used to make a connection. The source code can be found in the file JDBCConnect.java in the samples-dir\SQLAnywhere\JDBC directory. As presented, the example uses the JDBC 3.0 version of the iAnywhere JDBC driver to connect to the database. To use the jConnect 6.0.5 driver replace ianywhere.ml.jdbcodbc.jdbc3.IDriver with com.sybase.jdbc3.jdbc.SybDriver. You must also change the connection string if you want to use the jConnect 6.0.5 driver. Code alternatives are included as comments in the source code.

```java
import java.io.*;
import java.sql.*;

public class JDBCConnect
{
    public static void main( String args[] )
    {
        try
        {
            // Select the JDBC driver. May throw a SQLException.
            // Choices are jConnect 6.0 driver
            // or iAnywhere JDBC 3.0 driver.
            // Currently, we use the iAnywhere JDBC 3.0 driver.
            DriverManager.registerDriver( (Driver)
                Class.forName("com.sybase.jdbc3.jdbc.SybDriver").newInstance()
                "ianywhere.ml.jdbcodbc.jdbc3.IDriver").newInstance() );

            // Create a connection. Choices are TDS using jConnect,
            // Sun's JDBC-ODBC bridge, or the iAnywhere JDBC driver.
            // Currently, we use the iAnywhere JDBC driver.
            Connection con = DriverManager.getConnection(
                "jdbc:ianywhere:driver=SQL Anywhere 11;uid=DBA;pwd=sql" );

            // Create a statement object, the container for the SQL
```
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 JDBCCConnect.java:

- The java.io package contains the Sun Microsystems io classes, which are required for printing to the console window.
- The java.sql package contains the Sun Microsystems 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, JDBCCConnect.main is the only public method in the application.

The JDBCCConnect.main method carries out the following tasks:
1. Loads the iAnywhere JDBC 3.0 driver (identified by the driver string "ianywhere.ml.jdbcodbc.jdbc3.IDriver").

   Class.forName loads the driver. Using the newInstance method works around issues in some browsers.
2. Connects to the default running database using an iAnywhere JDBC driver URL. If you are using the jConnect driver instead, use the URL "jdbc:sysbase:Tds:localhost:2638" (as shown in comments) with "DBA" and "sql" as the user ID and password, respectively.

   DriverManager.getConnection 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**

**To create and execute the external connection example application**

1. At a command prompt, change to the samples-dir\SQLAnywhere\JDBC directory.

2. Start a database server with the sample database on your local computer using the following command:

   `dbeng11 samples-dir\demo.db`

3. Set the CLASSPATH environment variable.

   `set classpath=%classpath%;install-dir\java\jodbc.jar`

   If you are using the jConnect driver instead, then use the following (where path is your jConnect installation directory):

   `set classpath=path\jConnect-6_0\classes\jconn3.jar;%classpath%`

4. Run the following command to compile the example:

   `javac JDBCConnect.java`

5. Run the following command to run the example:

   `java JDBCConnect`

6. Confirm that a list of identification numbers with customer's names appears at the 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 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 source code in samples-dir\SQLAnywhere\JDBC\JDBCConnect.java.

```java
import java.io.*;
import java.sql.*;

public class JDBCConnect2
{
    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 driver manager does not need to be loaded. The following code has been removed from the example.

```java
DriverManager.registerDriver((Driver)
    Class.forName("com.sybase.jdbc3.jdbc.SybDriver").newInstance()
    "ianywhere.ml.jdbcodbc.IDriver").newInstance()
);
```

2. It connects to the default running database using the current connection. The URL in the getConnection call has been changed as follows:

```java
Connection con = DriverManager.getConnection("jdbc:default:connection");
```

3. The `System.exit()` statements have been removed.

Running the server-side connection example

**To create and execute the internal connection example application**

1. At a command prompt, change to the `samples-dir\SQLAnywhere\JDBC` directory.
2. Start a database server with the sample database on your local computer using the following command:

   ```bash
   dbeng11 samples-dir\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:

   ```bash
   javac JDBCConnect2.java
   ```

5. Install the class into the sample database using Interactive SQL. Run the following statement:

   ```sql
   INSTALL JAVA NEW FROM FILE 'samples-dir\SQLAnywhere\JDBC\JDBCConnect2.class'
   ```

   You can also install the class using Sybase Central. While connected to the sample database, open the Java subfolder under External Environments and choose File » New » Java Class. Then follow the instructions in the wizard.

6. Define a stored procedure named JDBCConnect that acts as a wrapper for the JDBCConnect2.main method in the class:

   ```sql
   CREATE PROCEDURE JDBCConnect()
   EXTERNAL NAME 'JDBCConnect2.main([Ljava/lang/String;);V'
   LANGUAGE JAVA;
   ```

7. Call the JDBCConnect2.main method as follows:

   ```java
   call JDBCConnect();
   ```
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 modification 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:

  ```java
  con.setAutoCommit( false );
  ```

  In this statement, con is the current connection object. You could also set autocommit to true.

- **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:

  ```java
  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.
Using JDBC to access data

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 519.

Preparing for the examples

Sample code

The code fragments in this section are taken from the complete class in samples-dir\SQLAnywhere\JDBC\JDBCExample.java.

To 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 (samples-dir represents the SQL Anywhere samples directory):

   ```sql
   INSTALL JAVA NEW FROM FILE 'samples-dir\SQLAnywhere\JDBC\JDBCExample.class'
   ```

   You can also install the class using Sybase Central. While connected to the sample database, open the Java subfolder under External Environments and choose File » New » Java Class. Follow the instructions in the wizard.

Inserts, updates, and deletes using JDBC

The Statement object executes static SQL statements. You execute SQL statements such as INSERT, UPDATE, and DELETE, which do not return result sets, using the executeUpdate method of the Statement object. Statements, such as CREATE TABLE and other data definition statements, can also be executed using executeUpdate.
When using the iAnywhere 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 cause a degradation of performance. This is because the iAnywhere 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.

If you do not want your application to use batched inserts to insert large data into a long binary or long varchar column, the default maximum field size of any batched insert column is 256K. If your application needs to batch inserts with more than 256K of column data, then a larger maximum field size must be specified in the Statement.setMaxFieldSize() method before the batch insert performed.

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.

```java
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();
    }
}
```

Source code available
This code fragment is part of the JDBCExample class included in the samples-dir\SQLAnywhere\JDBC directory.

Notes

- 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.

To 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 517.

3. Define a stored procedure named JDBCExample that acts as a wrapper for the JDBCExample.main method in the class:

```sql
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:

```java
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.

```sql
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:

```java
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.

```sql
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 prior to actual 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.

For more information about prepared statements, see “Preparing statements” on page 26.

**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:

```java
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();
    }
}
```

Source code available
This code fragment is part of the `JDBCExample` class included in the `samples-dir\SQLAnywhere\JDBC` directory.

Notes

- 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.

To 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 517.

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.

**Using prepared statements for wide inserts**

The PreparedStatement.addBatch() method is useful 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 Connection.prepareStatement() methods.

    ```java
    // 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 should then be executed using the PreparedStatement.executeUpdate() method.

It should be noted that only the PreparedStatement.addBatch() method is supported and that the
PreparedStatement.executeUpdate() method needs to be called to execute the batch. None of batch methods
for the Statement object (i.e. Statement.addBatch(), Statement.clearBatch(), Statement.executeBatch() ) are
supported since these methods are completely optional and not very useful. For such static batches, it is best
to call Statement.execute() or Statement.executeQuery() on a single string with the batched statements
wrapped inside a BEGIN...END.

Notes

- BLOB parameters are not supported in batches.
- String/Binary parameters are supported but the size of the string/binary parameter is an issue. By default
  the string/binary parameter is restricted to 255 characters or 510 bytes. The reason for the restriction is
due to the underlying ODBC protocol and will not be discussed here. For further information, it is best
to view the documentation on passing arrays of parameters in ODBC. If, however, an application needs
to pass larger string or binary parameters within a batch then an additional method, setBatchStringSize,
has been provided to increase the size of the string/binary parameter. Note that this method must be
called prior to the first addBatch() call. If the method is called after the first addBatch() call, then the
new size setting will be ignored. So, when calling this method to change the size of a string/binary
parameter, the application needs to know ahead of time what the maximum string or binary value for
that parameter will be.

To use the setBatchStringSize method, you must modify the "code" above as follows:
// You need to cast "stmt" to an IPreparedStatment object
// to change the size of string/binary parameters.
ianywhere.ml.jdbcodbc.IPreparedStatement _stmt =
    (ianywhere.ml.jdbcodbc.IPreparedStatement)stmt;

// Now, for example, change the size of string parameter 4
// from the default 255 characters to 300 characters.
// Note that string parameters are measured in "characters".
_stmt.setBatchStringSize( 4, 300 );

// Change the size of binary parameter 6
// from the default 510 bytes to 750 bytes.
// Note that binary parameters are measured in "bytes".
_stmt.setBatchStringSize( 6, 750 );

// loop to batch "n" sets of parameters
// where n should not be too large
for( i=0; i < n; i++ )
{
    // stmt is the 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();
}

The maximum string/binary size of a parameter should be modified with caution. If the maximum is set too high, then the additional memory allocation cost will probably override any performance gained from using the batch. Also, there is a good chance the application will not know ahead of time what the maximum string or binary value for a particular parameter is. The recommendation then is to not change the maximum string or binary size of a parameter, but rather to use the batch method until a string or binary value larger than the current/default maximum is encountered. The application can, at that point, call executeBatch() to execute the parameters that are currently batched, then call the regular set and executeUpdate() method to execute until the large string/binary parameters have been handled, then switch back into batch mode when smaller string/binary parameters are encountered.

## 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 calling SQL code. It uses three executeQuery statements to obtain three different result sets.

```java
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();
}

Source code available
This code fragment is part of the JDBCExample class included in the samples-dir\SQLAnywhere\JDBC directory.

Notes
- This server-side JDBC example connects to the default running database using the current connection using getConnection.
- The executeQuery methods return result sets.

To 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 517.
3. Define a stored procedure named JDBCResults that acts as a wrapper for the JDBCExample.Results method in the class:
   
   ```
   CREATE PROCEDURE JDBCResults()
   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. From the Tools menu, choose Options.
   b. Click SQL Anywhere.
   c. Click the Results tab.
   d. Set the value for Maximum Number Of Rows To Display to 5000.
   e. Select Show All Result Sets.
   f. Click OK.
5. Call the JDBCExample.Results method as follows:

```java
CALL JDBCResults();
```

6. Check each of the three results tabs, Result Set 1, Result Set 2, and Result Set 3.

**Miscellaneous JDBC notes**

- **Access permissions** Like all Java classes in the database, classes containing JDBC statements can be accessed by any user provided that 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.
Using 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

\{\{\textbf{keyword} \, \textbf{parameters}\}\}\}

In Interactive SQL, the braces \textit{must} be doubled. There must not be a space between successive braces: \texttt{"\{\{" is acceptable, but \texttt{"\{\{" 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 executed by Interactive SQL.

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 name of the current user in a database management system-neutral way, you would execute the following:

```sql
SELECT \{\{\text{FN USER()}\}\}
```

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 iAnywhere JDBC driver and by the jConnect driver.

### iAnywhere JDBC driver supported functions

<table>
<thead>
<tr>
<th>Numeric functions</th>
<th>String functions</th>
<th>System functions</th>
<th>Time/date functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ABS</em></td>
<td><em>ASCII</em></td>
<td><em>IFNULL</em></td>
<td><em>CURDATE</em></td>
</tr>
<tr>
<td><em>ACOS</em></td>
<td><em>CHAR</em></td>
<td><em>USERNAME</em></td>
<td><em>CURTIME</em></td>
</tr>
<tr>
<td><em>ASIN</em></td>
<td><em>CONCAT</em></td>
<td></td>
<td><em>DAYNAME</em></td>
</tr>
<tr>
<td><em>ATAN</em></td>
<td><em>DIFFERENCE</em></td>
<td></td>
<td><em>DAYOFMONTH</em></td>
</tr>
<tr>
<td><em>ATAN2</em></td>
<td><em>INSERT</em></td>
<td></td>
<td><em>DAYOFWEEK</em></td>
</tr>
<tr>
<td><em>CEILING</em></td>
<td><em>LCASE</em></td>
<td></td>
<td><em>DAYOFYEAR</em></td>
</tr>
<tr>
<td><em>COS</em></td>
<td><em>LEFT</em></td>
<td></td>
<td><em>HOUR</em></td>
</tr>
<tr>
<td><em>COT</em></td>
<td><em>LENGTH</em></td>
<td></td>
<td><em>MINUTE</em></td>
</tr>
<tr>
<td><em>DEGREES</em></td>
<td><em>LOCATE</em></td>
<td></td>
<td><em>MONTH</em></td>
</tr>
<tr>
<td><em>EXP</em></td>
<td><em>LOCATE_2</em></td>
<td></td>
<td><em>MONTHNAME</em></td>
</tr>
<tr>
<td><em>FLOOR</em></td>
<td><em>LTRIM</em></td>
<td></td>
<td><em>NOW</em></td>
</tr>
<tr>
<td><em>LOG</em></td>
<td><em>REPEAT</em></td>
<td></td>
<td><em>QUARTER</em></td>
</tr>
</tbody>
</table>
### Numeric functions
- **LOG10**
- **MOD**
- **PI**
- **POWER**
- **RADIANS**
- **RAND**
- **ROUND**
- **SIGN**
- **SIN**
- **SQRT**
- **TAN**
- **TRUNCATE**

### String functions
- **RIGHT**
- **RTRIM**
- **SOUNDEX**
- **SPACE**
- **SUBSTRING**
- **UCASE**

### System functions
- **SECOND**
- **WEEK**
- **YEAR**
- **SPACE**

### Time/date functions

---

### jConnect supported functions

<table>
<thead>
<tr>
<th>Numeric functions</th>
<th>String functions</th>
<th>System functions</th>
<th>Time/date functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABS</strong></td>
<td><strong>ASCII</strong></td>
<td><strong>DATABASE</strong></td>
<td><strong>CURDATE</strong></td>
</tr>
<tr>
<td><strong>ACOS</strong></td>
<td><strong>CHAR</strong></td>
<td><strong>IFNULL</strong></td>
<td><strong>CURTIME</strong></td>
</tr>
<tr>
<td><strong>ASIN</strong></td>
<td><strong>CONCAT</strong></td>
<td><strong>USER</strong></td>
<td><strong>DAYNAME</strong></td>
</tr>
<tr>
<td><strong>ATAN</strong></td>
<td><strong>DIFFERENCE</strong></td>
<td><strong>CONVERT</strong></td>
<td><strong>DAYOFMONTH</strong></td>
</tr>
<tr>
<td><strong>ATAN2</strong></td>
<td><strong>LCASE</strong></td>
<td></td>
<td><strong>DAYOFWEEK</strong></td>
</tr>
<tr>
<td><strong>CEILING</strong></td>
<td><strong>LENGTH</strong></td>
<td></td>
<td><strong>HOUR</strong></td>
</tr>
<tr>
<td><strong>COS</strong></td>
<td><strong>REPEAT</strong></td>
<td></td>
<td><strong>MINUTE</strong></td>
</tr>
<tr>
<td><strong>COT</strong></td>
<td><strong>RIGHT</strong></td>
<td></td>
<td><strong>MONTH</strong></td>
</tr>
<tr>
<td><strong>DEGREES</strong></td>
<td><strong>SOUNDEX</strong></td>
<td></td>
<td><strong>MONTHNAME</strong></td>
</tr>
<tr>
<td><strong>EXP</strong></td>
<td><strong>SPACE</strong></td>
<td></td>
<td><strong>NOW</strong></td>
</tr>
<tr>
<td>Numeric functions</td>
<td>String functions</td>
<td>System functions</td>
<td>Time/date functions</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>FLOOR</td>
<td>SUBSTRING</td>
<td></td>
<td>QUARTER</td>
</tr>
<tr>
<td>LOG</td>
<td>UCASE</td>
<td></td>
<td>SECOND</td>
</tr>
<tr>
<td>LOG10</td>
<td></td>
<td></td>
<td>TIMESTAMPADD</td>
</tr>
<tr>
<td>PI</td>
<td></td>
<td></td>
<td>TIMESTAMPDIFF</td>
</tr>
<tr>
<td>POWER</td>
<td></td>
<td></td>
<td>YEAR</td>
</tr>
<tr>
<td>RADIANS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

For example, to obtain database properties with the sa_db_info procedure using SQL escape syntax, you would execute the following in Interactive SQL:

```sql
CALL sa_db_info( 0 )
```
iAnywhere JDBC 3.0 API support

All mandatory classes and methods of the JDBC 3.0 specification are supported. 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 );
Introduction to embedded SQL

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 dblib11.dll.
- On Unix operating systems, the interface library is libdblib11.so, libdblib11.sl, or libdblib11.a, depending on the operating system.
- On Mac OS X, the interface library is libdblib11.dylib.

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 DLBLIB to form an executable file. When the database server is running, this executable file uses DLBLIB to interact with the database server. The database server does not have to be running when the program is preprocessed.

For Windows, there are separate import libraries for Microsoft Visual C++, for Watcom C/C++, and for Borland C++.

Under Windows, the use of import libraries is one method for developing applications that call functions in DLLs. SQL Anywhere also provides an alternative, and recommended method which avoids the use of import libraries. For more information, see “Loading DLBLIB dynamically under Windows” on page 536.

### Running the SQL preprocessor

The SQL preprocessor is an executable named sqlpp.

The SQLPP command line is as follows:

```
sqlpp [ options ] sql-filename [ output-filename ]
```

The SQL preprocessor processes a C program with embedded SQL before the C or C++ compiler is run. The preprocessor translates the SQL statements into C/C++ language source that is put into the output file. The normal extension for source programs with embedded SQL is .sqc. The default output file name is the sql-filename with an extension of .c. If the sql-filename already has a .c extension, then the output file name extension is .cc by default.

**Reprocessing embedded SQL**

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.

For a full listing of the command line options, see “SQL preprocessor” on page 590.

### Supported compilers

The C language SQL preprocessor has been used in conjunction with the following compilers:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Compiler</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Watcom C/C++</td>
<td>9.5 or later</td>
</tr>
<tr>
<td>Windows</td>
<td>Microsoft Visual C++</td>
<td>6.0 or later</td>
</tr>
<tr>
<td>Windows</td>
<td>Borland C++</td>
<td>4.5</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>Microsoft Visual C++</td>
<td>2005</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>Microsoft eMbedded Visual C++</td>
<td>3.0, 4.0</td>
</tr>
</tbody>
</table>
Operating system | Compiler | Version
--- | --- | ---
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.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlca.h</td>
<td>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.</td>
</tr>
<tr>
<td>sqlda.h</td>
<td>SQL Descriptor Area structure definition included in embedded SQL programs that use dynamic SQL.</td>
</tr>
<tr>
<td>sqldef.h</td>
<td>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.</td>
</tr>
<tr>
<td>sqlerr.h</td>
<td>Definitions for error codes returned in the sqlcode field of the SQLCA.</td>
</tr>
<tr>
<td>sqlstate.h</td>
<td>Definitions for ANSI/ISO SQL standard error states returned in the sqlstate field of the SQLCA.</td>
</tr>
<tr>
<td>pshpk1.h, pshpk4.h, poppk.h</td>
<td>These headers ensure that structure packing is handled correctly.</td>
</tr>
</tbody>
</table>

**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.

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.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Compiler</th>
<th>Import library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Microsoft Visual C++</td>
<td><code>dblihtm.lib</code></td>
</tr>
</tbody>
</table>
The `libdbtasks11` libraries are called by the `libdblib11` libraries. Some compilers locate `libdbtasks11` automatically. For others, you need to specify it explicitly.

## A simple example

The following is a very simple example of an embedded SQL program.

```c
#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 SQL and 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.

For a description of fetching data, see “Fetching data” on page 574.

**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.

```c
EXEC SQL INCLUDE SQLCA;
```

Every C program using embedded SQL must initialize a SQLCA first:

```c
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.

```c
db_fini( &sqlca );
```

**Loading DBLIB dynamically 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 SQL Anywhere applications. DBLIB can be loaded dynamically, without having to link against the import library, using the `esqdll.c` module in the `SDK\C` subdirectory of your installation directory.

A similar technique could be used to dynamically load DBLIB on Unix platforms.

**To load the 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> line 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.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SQL_OS_WINDOWS</td>
<td>All Windows operating systems</td>
</tr>
</tbody>
</table>

4. Compile esqldll.c.

5. Instead of linking against the imports 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 samples-dir \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 `samples-dir\SQLAnywhere\C` directory. For Windows Mobile, an additional example is located in the `samples-dir\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\v6.1
  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 samples-dir\SQLAnywhere\CE\esql_sample.

This will build the following examples.

- **CUR** An embedded SQL static cursor example
- **DCUR** An embedded SQL dynamic cursor example
- **ODBC** An ODBC example which is discussing in “ODBC samples” on page 471.

### Running the sample programs

The executable files and corresponding source code are located in the `samples-dir\SQLAnywhere\C` directory. For Windows Mobile, an additional example is located in the `samples-dir\SQLAnywhere\CE\esql_sample` directory.

#### To 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.

#### To 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 11 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.
Windows samples

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.

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. See “Dynamic cursor sample” on page 540.

For information about where the source code can be found and how to build this example program, see “Sample embedded SQL programs” on page 538.

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. Note that the fetch routine checks for warning conditions (such as Row not found) 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.

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. See “Static cursor sample” on page 540.

For information about where the source code can be found and how to build this sample program, see “Sample embedded SQL programs” on page 538.

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 of the form:

```
UID=DBA;PWD=sql;DBF=samples-dir\demo.db
```

The C program with the embedded SQL is held in the `samples-dir\SQLAnywhere\C` directory. For Windows Mobile, a dynamic cursor example is located in the `samples-dir\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.

### Size of the SQLDA

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.
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_UNSINSMALLINT**  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_UNSINT**  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).

  ```c
  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.

```c
typedef struct VARCHAR {
    unsigned short int 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.

```c
typedef struct NVARCHAR {
    unsigned short int len;
    char array[1];
} NVARCHAR;
```

DT_LONGVARCHAR  Long varying length character string, in the CHAR character set.

```c
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.

For more information, see “Sending and retrieving long values” on page 582.

DT_LONGNVARCHAR  Long varying length character string, in the NCHAR character set. The macro defines a structure, as follows:

```c
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.

For more information, see “Sending and retrieving long values” on page 582.

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 {
    unsigned short int len;
    char array[1];
} BINARY;

● **DT_LONGBINARY**  Long binary data. The macro defines a structure, as follows:

```c
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.

For more information, see “Sending and retrieving long values” on page 582.

● **DT_TIMESTAMP_STRUCT**  SQLDATETIME structure with fields for each part of a timestamp.

```c
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 a programmer 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.

See:

- “date_format option [database]” [SQL Anywhere Server - Database Administration]
- “date_order option [database]” [SQL Anywhere Server - Database Administration]
- “time_format option [compatibility]” [SQL Anywhere Server - Database Administration]
- “timestamp_format option [compatibility]” [SQL Anywhere Server - Database Administration]

● **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.

For more information, see “GET DATA statement [ESQL]” [SQL Anywhere Server - SQL Reference] and “SET statement” [SQL Anywhere Server - SQL Reference].
Using host variables

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.

You cannot use host variables in batches.

For information about dynamic SQL, see “Static and dynamic SQL” on page 561.

Declaring host variables

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);
```

For a more extensive example, see “Static cursor sample” on page 540.
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, FIXCHAR, NFIXCHAR, DATETIME (SQLDATETIME), BIT, BIGINT, or UNSIGNED BIGINT. They are used as follows:

```sql
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 a 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.

<table>
<thead>
<tr>
<th>C data type</th>
<th>Embedded SQL interface type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td><code>si;</code></td>
<td>DT_SMALLINT</td>
</tr>
<tr>
<td>short int</td>
<td><code>si;</code></td>
<td>DT_SMALLINT</td>
</tr>
<tr>
<td>unsigned short int</td>
<td><code>usi;</code></td>
<td>DT_UNS SMALLINT</td>
</tr>
<tr>
<td>long</td>
<td><code>l;</code></td>
<td>DT_INT</td>
</tr>
<tr>
<td>long int</td>
<td><code>l;</code></td>
<td>DT_INT</td>
</tr>
<tr>
<td>unsigned long int</td>
<td><code>ul;</code></td>
<td>DT_UNSINT</td>
</tr>
<tr>
<td>DECL_BIGINT</td>
<td><code>ll;</code></td>
<td>DT_BIGINT</td>
</tr>
<tr>
<td>DECL UNSIGNED_BIGINT</td>
<td><code>ull;</code></td>
<td>DT_UNSBIGINT</td>
</tr>
<tr>
<td>float</td>
<td><code>f;</code></td>
<td>DT_FLOAT</td>
</tr>
<tr>
<td>double</td>
<td><code>d;</code></td>
<td>DT_DOUBLE</td>
</tr>
<tr>
<td><strong>C data type</strong></td>
<td><strong>Embedded SQL interface type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>char a[n]; /<em>n&gt;=1</em>/</td>
<td>DT_STRING</td>
<td>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.</td>
</tr>
<tr>
<td>char *a;</td>
<td>DT_STRING</td>
<td>Null-terminated string, in CHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.</td>
</tr>
<tr>
<td>DECL_NCHAR a[n]; /<em>n&gt;=1</em>/</td>
<td>DT_NSTRING</td>
<td>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.</td>
</tr>
<tr>
<td>DECL_NCHAR *a;</td>
<td>DT_NSTRING</td>
<td>Null-terminated string, in NCHAR character set. This variable points to an area that can hold up to 32766 bytes plus the null terminator.</td>
</tr>
<tr>
<td>DECL_VARCHAR(n) a;</td>
<td>DT_VARCHAR</td>
<td>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).</td>
</tr>
<tr>
<td>DECL_NVARCHAR(n) a;</td>
<td>DT_NVARCHAR</td>
<td>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).</td>
</tr>
<tr>
<td>C data type</td>
<td>Embedded SQL interface type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DECL_LONGVARCHAR(n) a;</td>
<td>DT_LONGVARCHAR</td>
<td>Varying length long character string, in CHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.</td>
</tr>
<tr>
<td>DECL_LONGNVARCHAR(n) a;</td>
<td>DT_LONGNVARCHAR</td>
<td>Varying length long character string, in NCHAR character set, with three 4-byte length fields. Not null-terminated or blank-padded.</td>
</tr>
<tr>
<td>DECL_BINARY(n) a;</td>
<td>DT_BINARY</td>
<td>Varying length binary data with 2-byte length field. The maximum value for n is 32765 (bytes).</td>
</tr>
<tr>
<td>DECL_LONGBINARY(n) a;</td>
<td>DT_LONGBINARY</td>
<td>Varying length long binary data with three 4-byte length fields.</td>
</tr>
<tr>
<td>char a; /<em>n=1</em>/ DECL_FIXCHAR(n) a;</td>
<td>DT_FIXCHAR</td>
<td>Fixed length character string, in CHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).</td>
</tr>
<tr>
<td>DECL_NCHAR a; /<em>n=1</em>/ DECL_NFIXCHAR(n) a;</td>
<td>DT_NFIXCHAR</td>
<td>Fixed length character string, in NCHAR character set. Blank-padded but not null-terminated. The maximum value for n is 32767 (bytes).</td>
</tr>
<tr>
<td>DECL_DATETIME a;</td>
<td>DT_TIME-STAMP_STRUCT</td>
<td>SQLDATETIME structure</td>
</tr>
</tbody>
</table>

**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.

For more information about locales and character sets, see “Understanding locales” [SQL Anywhere Server - Database Administration].

For more information about changing the CHAR character set, see “CharSet connection parameter [CS]” [SQL Anywhere Server - Database Administration] or “db_change_char_charset function” on page 598.

For more information about changing the NCHAR character set, see “db_change_nchar_charset function” on page 598.

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.

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 a user ID, option name, or option value.

Host variables cannot be used in place of a table name or a column name in any statement.

**SQLCODE and SQLSTATE host variables**

The ISO/ANSI standard allows an embedded SQL source file to declare the following special host variables within a declaration section:

```c
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).

The SQLCODE and SQLSTATE host variables must be visible in the scope of every embedded SQL statement that generates database requests.

For more information, see the description of the sqlpp -k option in “SQL preprocessor” on page 590.

The following is valid embedded SQL:

```sql
EXEC SQL INCLUDE SQLCA;
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 ...}
```

The following is not valid embedded SQL:

```sql
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...
    // No SQLSTATE in scope of this statement
}
```

**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 short int 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.

### Using indicator variables to handle 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];
short int 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).

Using indicator variables 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.

Using indicator values for 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.

<table>
<thead>
<tr>
<th>Indicator value</th>
<th>Supplying value to database</th>
<th>Receiving value from database</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>Host variable value</td>
<td>Retrieved value was truncated—actual length in indicator variable</td>
</tr>
<tr>
<td>0</td>
<td>Host variable value</td>
<td>Fetch successful, or conversion_error set to On</td>
</tr>
<tr>
<td>-1</td>
<td>NULL value</td>
<td>NULL result</td>
</tr>
<tr>
<td>-2</td>
<td>NULL value</td>
<td>Conversion error (when conversion_error is set to Off only). SQLCODE indicates a CANNOT_CONVERT warning</td>
</tr>
<tr>
<td>Indicator value</td>
<td>Supplying value to database</td>
<td>Receiving value from database</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>&lt; -2</td>
<td>NULL value</td>
<td>NULL result</td>
</tr>
</tbody>
</table>

For more information about retrieving long values, see “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 imports 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 long integer that contains the length of the SQLCA structure (136 bytes).
- **sqlcode**: A long 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.

  For a full listing of error codes, see Error Messages.

- **sqlerrml**: The length of the information in the sqllerrmc 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, sqllerrmc contains the table name, which is inserted into the error message at the appropriate place.

  For a full listing of error messages, see Error Messages.

- **sqlerrp**: Reserved.
- **sqllerrd**: A utility array of long 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.

For more SQLSTATE values, see “SQL Anywhere error messages sorted by SQLSTATE” [Error Messages].

**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 statement** On a cursor OPEN, 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). In the case of a wide fetch, SQLCOUNT is the number of rows actually fetched, and is less than or equal to the number of rows requested. During a wide fetch, SQLE_NOTFOUND is only set if no rows are returned.

  For more information about wide fetches, see “Fetching more than one row at a time” on page 577.

  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.
In the WITH VARIABLE RESULT clause 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.

In the case of a syntax error, SQLSyntaxError, this 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.

---

### SQLCA management for multi-threaded or reentrant code

You can use embedded SQL statements in multi-threaded or reentrant code. However, if you use a single connection, you are restricted to one active request per connection. In a multi-threaded 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.

Any given database connection is accessible only from one SQLCA, with the exception of the cancel instruction, which must be issued from a separate thread.

For information about canceling requests, see “Implementing request management” on page 589.

The following is an example of multi-threaded embedded SQL reentrant code.

```c
#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++;
            }
        }
    }
}
Using multiple SQLCAs

To manage multiple SQLCAs in your application

1. You must not use the option on the SQL preprocessor that generates non-reentrant code (-r-). The reentrant code is a little larger and a little slower because statically initialized global variables cannot be used. However, these effects are minimal.

2. 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.

3. 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 
   ```sql
   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.

   For information about creating SQLCAs, see “SET SQLCA statement [ESQL]” in [SQL Anywhere Server - SQL Reference].

When to use multiple SQLCAs

You can use the multiple SQLCA support in any of the supported embedded SQL environments, but it is only required in reentrant code.

The following list details the environments where multiple SQLCAs must be used:

- **Multi-threaded applications** Each thread must have its own SQLCA. This can also happen when you have a DLL that uses embedded SQL and is called by more than one thread in your application.

- **Dynamic link libraries and shared libraries** A DLL has only one data segment. While the database server is processing a request from one application, it may yield to another application that makes a request to the database server. If your DLL uses the global SQLCA, both applications are using it at the same time. Each Windows application must have its own SQLCA.

- **A DLL with one data segment** A DLL can be created with only one data segment or one data segment for each application. If your DLL has only one data segment, you cannot use the global SQLCA for the same reason that a DLL cannot use the global SQLCA. Each application must have its own SQLCA.
Connection management with multiple SQLCAs

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 See “SET CONNECTION statement [Interactive SQL] [ESQL]” [SQL Anywhere Server - SQL Reference].

All operations on a given database connection must use the same SQLCA that was used when the connection was established.

**Record locking**
Operations on different connections are subject to the normal record locking mechanisms and may cause each other to block and possibly to deadlock. For information about locking, see “Using procedures, triggers, and batches” [SQL Anywhere Server - SQL Usage].
Static and dynamic SQL

There are two ways to embed SQL statements into a C program:

- Static statements
- Dynamic statements

Until now, static SQL has been discussed. This section compares static and dynamic SQL.

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. All examples to this point have used static embedded SQL statements. See “Using host variables” on page 546.

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.

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 place holders in the appropriate positions of the prepared statement.

For information about the SQLDA, see “The SQL descriptor area (SQLDA)” on page 565.

A place holder is put in the statement to indicate where host variables are to be accessed. A place holder 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 place holder indicating a reference to the SQL descriptor area.

A host variable used to pass information to the database is called a bind variable.

Example

For example:

```sql
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
char Street[30];
char City[20];
```
short int cityind;
long empnum;
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
    "UPDATE %s SET Street = :?, City = :?"
    "WHERE EmployeeID = :?",
    tablename );
EXEC SQL PREPARE S1 FROM :comm;
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;
/* 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

For a complete description of the SQLDA structure, see “The SQL descriptor area (SQLDA)” on page 565.

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.

For more information, see “Indicator variables” on page 551.

**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 to the C programmer, 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:

```sql
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;
```
Drop statements after use
To avoid consuming unnecessary resources, ensure that statements are dropped after use.

For a complete example using cursors for a dynamic select statement, see “Dynamic cursor sample” on page 540.

For more information about the functions mentioned above, see “Library function reference” on page 593.
The SQL descriptor area (SQLDA)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure passes 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 library or DLL that you can use to manage SQLDAs. For descriptions, see “Library function reference” on page 593.

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.

The SQLDA header file

The contents of `sqlda.h` are as follows:

```c
#ifndef _SQLDA_H_INCLUDED
#define _SQLDA_H_INCLUDED
#define II_SQLDA
#include "sqlca.h"
#if defined( _SQL_PACK_STRUCTURES )
#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 */
  short int sqllen; /* length of host variable */
  void      *sqldata; /* address of variable */
  short int *sqlind; /* indicator variable pointer */
  struct sqlname sqlname;
};
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 */
};
typedef struct sqlda    SQLDA;
typedef struct sqlvar   SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname  SQLNAME, SQLDA_NAME;
#endif SQLDASIZE
#define SQLDASIZE(n)    ( sizeof( struct sqlda ) + \
                      (n-1) * sizeof( struct sqlvar) )
```
SQLDA fields

The SQLDA fields have the following meanings:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqldaid</td>
<td>An 8-byte character field that contains the string SQLDA as an identification of the SQLDA structure. This field helps in debugging when you are looking at memory contents.</td>
</tr>
<tr>
<td>sqldabc</td>
<td>A long integer containing the length of the SQLDA structure.</td>
</tr>
<tr>
<td>sqln</td>
<td>The number of variable descriptors allocated in the sqlvar array.</td>
</tr>
<tr>
<td>sqld</td>
<td>The number of variable descriptors that are valid (contain information describing a host variable). This field is set by the DESCRIBE statement and sometimes by the programmer when supplying data to the database server.</td>
</tr>
<tr>
<td>sqlvar</td>
<td>An array of descriptors of type struct sqlvar, each describing a host variable.</td>
</tr>
</tbody>
</table>

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. See “Embedded SQL data types” on page 542.
  
  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. 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.
  
  For more information about the length field, see “SQLDA sqllen field values” on page 568.
The SQL descriptor area (SQLDA)

- **sqldata**  A pointer to the memory occupied by this variable. This memory must correspond to the sqltype and sqllen fields.

For storage formats, see “Embedded SQL data types” on page 542.

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 is a short int. 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. See “conversion_error option [compatibility]” [SQL Anywhere Server - Database Administration].

For more information, see “Indicator variables” on page 551.

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:

```c
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.
SQLDA sqllen field values

The sqllen field length of the sqlvar structure in a SQLDA is used in the following kinds of interactions with
the database server:

- **describing values**  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. See
  “Describing values” on page 568.
- **retrieving values**  Retrieving values from the database. See “Retrieving values” on page 571.
- **sending values**  Sending information to the database. See “Sending values” on page 570.

These interactions are described in this section.

The following tables detail each of these interactions. These tables list the interface constant types (the
DT_ types) found in the sqldef.h header file. These constants would be placed in the SQLDA sqltype field.

For information about sqltype field values, see “Embedded SQL data types” on page 542.

In static SQL, a SQLDA is still used, but it is generated and completely filled in by the SQL preprocessor.
In this static case, the tables give the correspondence between the static C language host variable types and
the interface constants.

### Describing values

The following table indicates the values of the sqllen and sqltype structure members returned by the
DESCRIBE statement for the various database types (both SELECT LIST and BIND VARIABLE
DESCRIBE statements). In the case of 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 sqllen fields. The embedded SQL type is obtained by a bitwise AND
of sqltype with DT_TYPES (sqltype & DT_TYPES).

For information about embedded SQL data types, see “Embedded SQL data types” on page 542.

<table>
<thead>
<tr>
<th>Database field type</th>
<th>Embedded SQL type returned</th>
<th>Length (in bytes) returned on describe</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>DT_BIGINT</td>
<td>8</td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>DT_BINARY</td>
<td>n</td>
</tr>
<tr>
<td>BIT</td>
<td>DT_BIT</td>
<td>1</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>DT_FIXCHAR</td>
<td>n</td>
</tr>
<tr>
<td>DATE</td>
<td>DT_DATE</td>
<td>length of longest formatted string</td>
</tr>
<tr>
<td>Database field type</td>
<td>Embedded SQL type returned</td>
<td>Length (in bytes) returned on describe</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>DT_DECIMAL</td>
<td>high byte of length field in SQLDA set to p, and low byte set to s</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DT_DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DT_FLOAT</td>
<td>4</td>
</tr>
<tr>
<td>INT</td>
<td>DT_INT</td>
<td>4</td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>DT_LONG_BINARY</td>
<td>32767</td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>DT_LONGNVARCHAR(^1)</td>
<td>32767</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>DT_LONGVARCHAR</td>
<td>32767</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>DT_NFIXCHAR(^1)</td>
<td>n times maximum character length in client's NCHAR character set</td>
</tr>
<tr>
<td>NVARCHAR(n)</td>
<td>DT_NVARCHAR(^1)</td>
<td>n times maximum character length in client's NCHAR character set</td>
</tr>
<tr>
<td>REAL</td>
<td>DT_FLOAT</td>
<td>4</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>DT_SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>TIME</td>
<td>DT_TIME</td>
<td>length of longest formatted string</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DT_TIMESTAMP</td>
<td>length of longest formatted string</td>
</tr>
<tr>
<td>TINYINT</td>
<td>DT_TINYINT</td>
<td>1</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>DT_UNSBIGINT</td>
<td>8</td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>DT_UNSINT</td>
<td>4</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>DT_UNSSMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>DT_VARCHAR</td>
<td>n</td>
</tr>
</tbody>
</table>

\(^1\) In embedded SQL, NCHAR, NVARCHAR, and LONG NVARCHAR are described as either DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, the types are described as DT_NFIXCHAR, DT_NVARCHAR, and DT_LONGNVARCHAR, respectively. See “db_change_nchar_charset function” on page 598.
## 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 format.

<table>
<thead>
<tr>
<th>Embedded SQL data type</th>
<th>Program action to set the length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_BIGINT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_BINARY(n)</td>
<td>Length taken from field in BINARY structure.</td>
</tr>
<tr>
<td>DT_BIT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_DATE</td>
<td>Length determined by terminating \0.</td>
</tr>
<tr>
<td>DT_DOUBLE</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_FIXCHAR(n)</td>
<td>Length field in SQLDA determines length of string.</td>
</tr>
<tr>
<td>DT_FLOAT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_INT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_LONGBINARY</td>
<td>Length field ignored. See “Sending LONG data” on page 584.</td>
</tr>
<tr>
<td>DT_LONGNVARCHAR</td>
<td>Length field ignored. See “Sending LONG data” on page 584.</td>
</tr>
<tr>
<td>DT_LONGVARCHAR</td>
<td>Length field ignored. See “Sending LONG data” on page 584.</td>
</tr>
<tr>
<td>DT_NFIXCHAR(n)</td>
<td>Length field in SQLDA determines length of string.</td>
</tr>
<tr>
<td>DT_NSTRING</td>
<td>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).</td>
</tr>
<tr>
<td>DT_NVARCHAR</td>
<td>Length taken from field in NVARCHAR structure.</td>
</tr>
<tr>
<td>DT_SMALLINT</td>
<td>No action required.</td>
</tr>
</tbody>
</table>
### Embedded SQL data type

<table>
<thead>
<tr>
<th>Embedded SQL data type</th>
<th>Program action to set the length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_STRING</td>
<td>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).</td>
</tr>
<tr>
<td>DT_TIME</td>
<td>Length determined by terminating \0.</td>
</tr>
<tr>
<td>DT_TIMESTAMP</td>
<td>Length determined by terminating \0.</td>
</tr>
<tr>
<td>DT_TIMESTAMP_STRUCT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSBIGINT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSINT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSSMALLINT</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_VARCHAR(n)</td>
<td>Length taken from field in VARCHAR structure.</td>
</tr>
<tr>
<td>DT_VARIABLE</td>
<td>Length determined by terminating \0.</td>
</tr>
</tbody>
</table>

### Retrieving values

Retrieving values

The following table indicates the values of the length field when you retrieve data from the database using a SQLDA. The sqllen 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.

<table>
<thead>
<tr>
<th>Embedded SQL data type</th>
<th>What the program must set length field to when receiving</th>
<th>How the database returns length information after fetching a value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_BIGINT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_BINARY(n)</td>
<td>Maximum length of BINARY structure (n+2). The maximum value for n is 32765.</td>
<td>len field of BINARY structure set to actual length in bytes.</td>
</tr>
<tr>
<td>DT_BIT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_DATE</td>
<td>Length of buffer.</td>
<td>\0 at end of string.</td>
</tr>
<tr>
<td>DT_DOUBLE</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>Embedded SQL data type</td>
<td>What the program must set length field to when receiving</td>
<td>How the database returns length information after fetching a value</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>DT_FIXCHAR(n)</td>
<td>Length of buffer, in bytes. The maximum value for n is 32767.</td>
<td>Padded with blanks to length of buffer.</td>
</tr>
<tr>
<td>DT_FLOAT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_INT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_LONGBINARY</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
</tr>
<tr>
<td>DT_LONGNVARCHAR</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
</tr>
<tr>
<td>DT_LONGVARCHAR</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
<td>Length field ignored. See “Retrieving LONG data” on page 583.</td>
</tr>
<tr>
<td>DT_NFIXCHAR(n)</td>
<td>Length of buffer, in bytes. The maximum value for n is 32767.</td>
<td>Padded with blanks to length of buffer.</td>
</tr>
<tr>
<td>DT_NSTRING</td>
<td>Length of buffer.</td>
<td>\0 at end of string.</td>
</tr>
<tr>
<td>DT_NVARCHAR(n)</td>
<td>Maximum length of NVARCHAR structure (n+2). The maximum value for n is 32765.</td>
<td>len field of NVARCHAR structure set to actual length in bytes of string.</td>
</tr>
<tr>
<td>DT_SMALLINT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_STRING</td>
<td>Length of buffer.</td>
<td>\0 at end of string.</td>
</tr>
<tr>
<td>DT_TIME</td>
<td>Length of buffer.</td>
<td>\0 at end of string.</td>
</tr>
<tr>
<td>DT_TIMESTAMP</td>
<td>Length of buffer.</td>
<td>\0 at end of string.</td>
</tr>
<tr>
<td>DT_TIMESTAMP_STRUCT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSBIGINT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSINT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>DT_UNSSMALLINT</td>
<td>No action required.</td>
<td>No action required.</td>
</tr>
<tr>
<td>Embedded SQL data type</td>
<td>What the program must set length field to when receiving</td>
<td>How the database returns length information after fetching a value</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>DT_VARCHAR(n)</td>
<td>Maximum length of VARCHAR structure (n+2). The maximum value for n is 32765.</td>
<td>len field of VARCHAR structure set to actual length in bytes of string.</td>
</tr>
</tbody>
</table>
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. See “SELECT statements that return at most one row” on page 574.

- **The SELECT statement may return multiple rows**  Use cursors to manage the rows of the result set. See “Using cursors in embedded SQL” on page 575.

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, a Row Not Found warning is returned. Errors and warnings are returned in the SQLCA structure. See “The SQL Communication Area (SQLCA)” on page 555.

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.

```sql
EXEC SQL BEGIN DECLARE SECTION;
long   ID;
char   name[41];
char   Sex;
char   birthdate[15];
short int ind_birthdate;
EXEC SQL END DECLARE SECTION;
...
int find_employee( long Employees )
{
   ID = Employees;
   EXEC SQL SELECT GivenName ||
      ' ' || Surname, Sex, BirthDate
   INTO :name, :Sex,
      :birthdate:ind_birthdate
   FROM Employees
   WHERE EmployeeID = :ID;
   if( SQLCODE == SQLE_NOTFOUND )
   {
      return( 0 ); /* Employees not found */
   }
   else if( SQLCODE < 0 )
   {
      return( -1 ); /* error */
   }
   else
```
Using 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.

For an introduction to cursors, see “Working with cursors” on page 32.

To manage a cursor in embedded SQL

1. Declare a cursor for a particular SELECT statement, using the DECLARE 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. See “The SQL Communication Area (SQLCA)” on page 555.
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:

```sql
void print_employees( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char name[50];
    char Sex;
    char birthdate[15];
    short int 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;

For complete examples using cursors, see “Static cursor sample” on page 540 and “Dynamic cursor sample” on page 540.

Cursor positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row

```
<table>
<thead>
<tr>
<th>Absolute row from start</th>
<th>Absolute row from end</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>−n − 1</td>
</tr>
<tr>
<td>1</td>
<td>−n</td>
</tr>
<tr>
<td>2</td>
<td>−n + 1</td>
</tr>
<tr>
<td>3</td>
<td>−n + 2</td>
</tr>
<tr>
<td>n − 2</td>
<td>−3</td>
</tr>
<tr>
<td>n − 1</td>
<td>−2</td>
</tr>
<tr>
<td>n</td>
<td>−1</td>
</tr>
<tr>
<td>n + 1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

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. See “FETCH statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].

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. See “PUT statement [ESQL]” [SQL Anywhere Server - SQL Reference].

**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. In some cases, the inserted row does not appear at all 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.

For a description, see “Use work tables in query processing (use All-rows optimization goal)” [SQL Anywhere Server - SQL Usage].

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).

**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. See “PUT statement [ESQL]” [SQL Anywhere Server - SQL Reference] and “EXECUTE statement [ESQL]” [SQL Anywhere Server - SQL Reference].

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 one 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 samples-dir \SQLAnywhere\esqlwidefetch\widefetch.sqc.
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLError();
    goto err; }

static void PrintSQLError()
{
    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->sqln;
    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->sqlid = 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].sqllen =
                sqlda->sqlvar[col].sqllen;
            // 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:
", 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 SQL_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 prior to 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.
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 (sqldata, sqllen, 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.

For information about how to retrieve and send values, see “Retrieving LONG data” on page 583, and “Sending LONG data” on page 584.

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:

```c
#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 structures are as follows:

```c
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. For background information, see “Sending and retrieving long values” on page 582.

The procedures are different depending on whether you are using static or dynamic SQL.

### To 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.
   
     For more information, see “Indicator variables” on page 551.
   - **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.

### To 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:

```c
#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
    // (sqllen is unused with DT_LONG types)
    sqlda->sqlld = 1; // using 1 sqlvar
    sqlvar = &sqlda->sqlvar[0];
    sqlvar->sqltype = DT_LONGVARCHAR;
    sqlvar->sqldata = longptr;

    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, ", 
            longptr->stored_len, 
            longptr->untrunc_len, 
            longptr->array[0], 
            longptr->array[DATA_LEN-1] );
    free_sqlda( sqlda );
    free( longptr );
}
```

### Sending LONG data

This section describes how to send LONG values to the database from embedded SQL applications. For background information, see “Sending and retrieving long values” on page 582.

The procedures are different depending on whether you are using static or dynamic SQL.
To 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.
   
   For more information, see “Indicator variables” on page 551.

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;
}
```

To 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), LONGVARCHARSIZE(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.
Using simple stored procedures

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:

```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.

```sql
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;
```

For more information, see “EXECUTE statement [ESQL]” [SQL Anywhere Server - SQL Reference], and “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 SQL_ERROR_PROCEDURE_COMPLETE or it returns SQL_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.

For a description of dynamic cursors, see “Dynamic SELECT statement” on page 563.

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).

For a complete description, see “DESCRIBE statement [ESQL]” [SQL Anywhere Server - SQL Reference].

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.
Embedded SQL programming techniques

This section contains a set of tips for developers of embedded SQL programs.

Implementing request management

The default behavior of the interface DLL is for applications to wait for completion of each database request before carrying out other functions. This behavior can be changed using request management functions. For example, when using Interactive SQL, the operating system is still active while Interactive SQL is waiting for a response from the database and Interactive SQL carries out some tasks in that time.

You can achieve application activity while a database request is in progress by providing a callback function. In this callback function, you must not do another database request except db_cancel_request. You can use the db_is_working function in your message handlers to determine if you have a database request in progress.

The db_register_a_callback function is used to register your application callback functions.

See also

- “db_register_a_callback function” on page 604
- “db_cancel_request function” on page 597
- “db_is_working function” on page 601

Backup functions

The db_backup function provides support for online backup in embedded SQL applications. The backup utility makes use of this function. You should only need to write a program to use this function if your backup requirements are not satisfied by the SQL Anywhere backup utility.

**BACKUP statement is recommended**

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. See “BACKUP statement” [SQL Anywhere Server - SQL Reference].

You can also access the backup utility directly using the Database Tools DBBackup function. See “DBBackup function” on page 957.

See also

- “db_backup function” on page 593
SQL preprocessor

The SQL preprocessor processes a C or C++ program containing embedded SQL before the compiler is run.

**Syntax**

```
sqlpp [ options ] input-file [ output-file ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td>Generate code that reduces data space size. Data structures are reused and initialized at execution time before use. This increases code size.</td>
</tr>
<tr>
<td>-e level</td>
<td>Flag as an error any static embedded SQL that is not part of a specified standard. The <code>level</code> value indicates the standard to use. For example, <code>sqlpp -e c03 ...</code> flags any syntax that is not part of the core SQL/2003 standard. The supported <code>level</code> values are:</td>
</tr>
<tr>
<td></td>
<td>● <code>c03</code> Flag syntax that is not core SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>p03</code> Flag syntax that is not full SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>c99</code> Flag syntax that is not core SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>p99</code> Flag syntax that is not full SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>e92</code> Flag syntax that is not entry-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>i92</code> Flag syntax that is not intermediate-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>f92</code> Flag syntax that is not full-SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>t</code> Flag non-standard host variable types</td>
</tr>
<tr>
<td></td>
<td>● <code>u</code> Flag syntax that is not supported by UltraLite</td>
</tr>
</tbody>
</table>

For compatibility with previous SQL Anywhere versions, you can also specify `e`, `i`, and `f`, which correspond to `e92`, `i92`, and `f92`, respectively.

<p>| -h width | Limit the maximum length of lines output by sqlpp to <code>width</code>. The continuation character is a backslash (<code>\</code>) and the minimum value of <code>width</code> is 10. |
| -k       | Notify the preprocessor that the program to be compiled includes a user declaration of SQLCODE. The definition must be of type LONG, but does not need to be in a declaration section. |
| -n       | Generate line number information in the C file. This consists of <code>#line</code> directives in the appropriate places in the generated C code. If the compiler that you are using supports the <code>#line</code> 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 <code>#line</code> directives are used indirectly by the source level debugger so that you can debug while viewing the SQC source file. |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o operating-system</td>
<td>Specify the target operating system. The supported operating systems are:</td>
</tr>
<tr>
<td></td>
<td>● WINDOWS Microsoft Windows</td>
</tr>
<tr>
<td></td>
<td>● UNIX Use this option if you are creating a 32-bit Unix application.</td>
</tr>
<tr>
<td></td>
<td>● UNIX64 Use this option if you are creating a 64-bit Unix application.</td>
</tr>
<tr>
<td>-q</td>
<td>Quiet mode—do not print messages.</td>
</tr>
<tr>
<td>-r</td>
<td>Generate non-reentrant code. For more information about reentrant code, see “SQLCA management for multi-threaded or reentrant code” on page 557.</td>
</tr>
<tr>
<td>-s len</td>
<td>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.</td>
</tr>
<tr>
<td>-u</td>
<td>Generate code for UltraLite. For more information, see “Embedded SQL API reference” [UltraLite - C and C++ Programming].</td>
</tr>
<tr>
<td>-w level</td>
<td>Flag as a warning any static embedded SQL that is not part of a specified standard. The level value indicates the standard to use. For example, sqlpp -w c03 ... flags any SQL syntax that is not part of the core SQL/2003 syntax. The supported level values are:</td>
</tr>
<tr>
<td></td>
<td>● c03 Flag syntax that is not core SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● p03 Flag syntax that is not full SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● c99 Flag syntax that is not core SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● p99 Flag syntax that is not full SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● e92 Flag syntax that is not entry-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● i92 Flag syntax that is not intermediate-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● f92 Flag syntax that is not full-SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● t Flag non-standard host variable types</td>
</tr>
<tr>
<td></td>
<td>● u Flag syntax that is not supported by UltraLite</td>
</tr>
<tr>
<td></td>
<td>For compatibility with previous SQL Anywhere versions, you can also specify e, i, and f, which correspond to e92, i92, and f92, respectively.</td>
</tr>
<tr>
<td>-x</td>
<td>Change multibyte strings to escape sequences so that they can pass through compilers.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>-z cs</code></td>
<td>Specify the collation sequence. For a list of recommended collation sequences, enter <code>dbinit -l</code> at a command prompt. 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 <code>-z</code> is not specified, the preprocessor attempts to determine a reasonable collation to use based on the operating system and the <code>SALANG</code> and <code>SACHARSET</code> environment variables. See “SACHARSET environment variable” [SQL Anywhere Server - Database Administration], and “SALANG environment variable” [SQL Anywhere Server - Database Administration].</td>
</tr>
</tbody>
</table>

| input-file | A C or C++ program containing embedded SQL to be processed. |
| output-file | The C language source file created by the SQL preprocessor. |

**Description**

The SQL preprocessor translates the SQL statements in the `input-file` into C language source that is put into the `output-file`. The normal extension for source programs with embedded SQL is `.sqc`. The default output file name is the `input-file` with an extension of `.c`. If `input-file` has a `.c` extension, the default output file name extension is `.cc`.

**See also**

- “Introduction to embedded SQL” on page 532
- “sql_flagger_error_level option [compatibility]” [SQL Anywhere Server - Database Administration]
- “sql_flagger_warning_level option [compatibility]” [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]
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_sqlnda function

Prototype

```c
struct sqlda * alloc_sqlnda( unsigned numvar);
```

Description

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_sqlnda_noind function.

alloc_sqlnda_noind function

Prototype

```c
struct sqlda * alloc_sqlnda_noind( unsigned numvar);
```

Description

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.

db_backup function

Prototype

```c
void db_backup( SQLCA * sqlca,
    int op,
    int file_num,
```
unsigned long page_num,
struct sqlda * sqlda);

Authorization

Must be connected as a user with DBA authority, REMOTE DBA authority (SQL Remote), or BACKUP authority.

Description

**BACKUP statement is recommended**

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. See “BACKUP statement” [SQL Anywhere Server - SQL Reference].


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.

  **Application must save buffer**

  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 SQLE_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 SQLE_NOERROR and then write the page. Continue reading the pages until you receive the SQLE_NOTFOUND condition.

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 as not to slow the server down with too many requests.

When you receive the SQLE_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 sqlda->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 SQLE_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 SQLE_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 sqlda 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 sqlda 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 SQLE_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 N lets the server know that the client is willing to accept at most N database pages at a time from the server. The server may return blocks of pages of less than size N if it is unable to allocate enough memory for blocks of N 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 \( N \) pages (specified in the DB_BACKUP_START_PARALLEL operation, or in a DB_BACKUP_INFO operation). For more information about this data type, see DT_LONGBINARY in “Embedded SQL data types” on page 542.

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.

```c
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 */
    /* SQLEIOESTIMATE 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
```

### db_cancel_request function

**Prototype**

```c
int db_cancel_request( SQLCA * sqlca );
```

**Description**

Cancels the currently active database server request. This function checks to make sure a database server request is active before sending the cancel request. If the function returns 1, then the cancel request was sent; if it returns 0, then no request was sent.
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.

**db_change_char_charset function**

**Prototype**

```c
unsigned int db_change_char_charset(
    SQLCA * sqlca,
    char * charset );
```

**Description**

Changes the application's CHAR character set for this connection. Data sent and fetched using FIXCHAR, VARCHAR, LONGVARCHAR, and STRING types are in the CHAR character set.

Returns 1 if the change is successful, 0 otherwise.

For a list of recommended character sets, see “Recommended character sets and collations” [SQL Anywhere Server - Database Administration].

**db_change_nchar_charset function**

**Prototype**

```c
unsigned int db_change_nchar_charset(
    SQLCA * sqlca,
    char * charset );
```

**Description**

Changes the application's NCHAR character set for this connection. Data sent and fetched using NFIXCHAR, NVARCHAR, LONGNVARCHAR, and 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.

Returns 1 if the change is successful, 0 otherwise.
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.

For a list of recommended character sets, see “Recommended character sets and collations” [SQL Anywhere Server - Database Administration].

**db_delete_file function**

**Prototype**
```c
void db_delete_file(
    SQLCA * sqlca,
    char * filename);
```

**Authorization**
Must be connected to a user ID with DBA authority or REMOTE DBA authority (SQL Remote).

**Description**
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. See DB_BACKUP_READ_RENAME_LOG in “db_backup function” on page 593.

You must be connected to a user ID with DBA authority.

**db_find_engine function**

**Prototype**
```c
unsigned short db_find_engine(
    SQLCA * sqlca,
    char * name);
```

**Description**
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.

### db_fini function

**Prototype**

```c
int db_fini( SQLCA * sqlca );
```

**Description**

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.

For information about using db_init in UltraLite applications, see “db_fini function” [*UltraLite - C and C++ Programming*].

### db_get_property function

**Prototype**

```c
unsigned int db_get_property(
    SQLCA * sqlca,
    a_db_property property,
    char * value_buffer,
    int value_buffer_size );
```

**Description**

This function is used to obtain information about the database interface or the server to which you are connected.

The arguments are as follows:

- **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 the terminating null character.

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, "11.0.0.1297").

Returns 1 if successful, 0 otherwise.

**db_init function**

**Prototype**

```
int db_init( SQLCA * sqlca );
```

**Description**

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.

Returns 1 if successful, 0 otherwise.

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.

In most cases, 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 more information, see “SQLCA management for multi-threaded or reentrant code” on page 557.

For information about using db_init in UltraLite applications, see “db_init function” [UltraLite - C and C++ Programming].

**db_is_working function**

**Prototype**

```
unsigned short db_is_working( SQLCA * sqlca );
```
Description

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.

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.

db_locate_servers function

Prototype

unsigned int db_locate_servers(
    SQLCA * sqlca,
    SQL_CALLBACK_PARM callback_address,
    void * callback_user_data);

Description

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.

Returns 1 if successful, 0 otherwise.
db_locate_servers_ex function

Prototype

```c
unsigned int db_locate_servers_ex(
    SQLCA * sqlca,
    SQL_CALLBACK_PARM callback_address,
    void * callback_user_data,
    unsigned int bitmask);
```

Description

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:

```c
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:

```c
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).

Returns 1 if successful, 0 otherwise.

For more information, see “Server Enumeration utility (dblocate)” [SQL Anywhere Server - Database Administration].

### db_register_a_callback function

**Prototype**

```c
void db_register_a_callback(
    SQLCA * sqlca,
    a_db_callback_index index,
    ( SQL_CALLBACK_PARM ) callback);
```

**Description**

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. See “MESSAGE statement” [SQL Anywhere Server - SQL Reference].

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 theLogFile 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:

```c
void SQL_CALLBACK debug_message_callback(
    SQLCA * sqlca,
    char * message_string);
```
For more information, see “LogFile connection parameter [LOG]” [SQL Anywhere Server - Database Administration].

- **DB_CALLBACK_START** The prototype is as follows:
  
  ```c
  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:
  
  ```c
  void SQL_CALLBACK finish_callback( SQLCA * sqlca );
  ```
  
  This function is called after the response to a database request has been received by the interface DLL. DB_CALLBACK_FINISH is used only on Windows operating systems.

- **DB_CALLBACK_CONN_DROPPED** The prototype is as follows:
  
  ```c
  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:
  
  ```c
  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:

  ```c
  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.

  The callback prototype is as follows:

  ```c
  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 and you may want to handle different message types in different ways. The available message types are MESSAGE_TYPE_INFO, MESSAGE_TYPE_WARNING, MESSAGE_TYPE_ACTION, and MESSAGE_TYPE_STATUS.
These constants are defined in sqldef.h. 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.

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:

```c
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.
db_start_database function

Prototype

unsigned int db_start_database( SQLCA * sqlca, char * parms );

Arguments

- sqlca A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.

- 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"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

Description

The database is started on an existing server, if possible. Otherwise, a new server is started. The steps carried out to start a database are described in “Locating a database server” [SQL Anywhere Server - Database Administration].

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. For information, see “-gd server option” [SQL Anywhere Server - Database Administration].

db_start_engine function

Prototype

unsigned int db_start_engine( SQLCA * sqlca, char * parms );

Arguments

- sqlca A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.

- 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"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].
**Description**

Starts the database server if it is not running.

For a description of the steps carried out by this function, see “Locating a database server” [SQL Anywhere Server - Database Administration].

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 and names it demo, but does not load the database, despite the DBF connection parameter:

```c
db_start_engine( &sqlca,
                 "DBF=samples-dir\demo.db;START=dbeng11" );
```

If you want to start a database and the server, include the database file in the StartLine (START) connection parameter:

```c
db_start_engine( &sqlca,
                 "ENG=eng_name;START=dbeng11 samples-dir\demo.db" );
```

This call starts the server, names it eng_name, and starts the SQL Anywhere sample database on that server.

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.

The ForceStart (FORCE) connection parameter is used only by the `db_start_engine` function. When 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:

   ```
   start dbeng11 -n server_1 demo.db
   ```

2. Force a new server to start and connect to it:

   ```
   db_start_engine( &sqlda,
                   "START=dbeng11 -n server_2 mydb.db;ForceStart=YES" )
   ```

If ForceStart (FORCE) was not used, and without a ServerName (ENG) parameter, 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.

**db_stop_database function**

**Prototype**

```c
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

**Arguments**

- `sqlca` A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.
● **parms**  A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example:

"UID=DA;PWD=sql;DBF=c:\\db\\mydatabase.db"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

**Description**

Stop the database identified by DatabaseName (DBN) on the server identified by ServerName (ENG). 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 is 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. For information, see “-gd server option” [SQL Anywhere Server - Database Administration].

---

**db_stop_engine function**

**Prototype**

```c
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

**Arguments**

- **sqlca**  A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.

- **parms**  A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form KEYWORD=value. For example,

"UID=DA;PWD=sql;DBF=c:\\db\\mydatabase.db"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

**Description**

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 (ENG) 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 Unconditional is yes, 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 “-gk server option” [SQL Anywhere Server - Database Administration].

### db_string_connect function

**Prototype**

```c
unsigned int db_string_connect( SQLCA * sqlca, char * parms);
```

**Arguments**

- **sqlca** A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.

- **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"
```

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

**Description**

Provides extra functionality beyond the embedded SQL CONNECT statement.

For a description of the algorithm used by this function, see “Troubleshooting connections” [SQL Anywhere Server - Database Administration].

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.

### db_string_disconnect function

**Prototype**

```c
unsigned int db_string_disconnect( SQLCA * sqlca, char * parms);
```

**Arguments**

- **sqlca** A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.
• **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"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

**Description**

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 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.

**db_string_ping_server function**

**Prototype**

```c
unsigned int db_string_ping_server(
    SQLCA * sqlca,
    char * connect_string,
    unsigned int connect_to_db);
```

**Description**

- **connect_string**  The connect_string is a normal connect 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 connect string is sufficient to connect to the named database on the named server.

- **connect_to_db**  If connect_to_db is zero, then the function only attempts to locate a server. It returns TRUE only if the connect string is sufficient to locate a server. It makes no attempt to connect to the database.

**db_time_change function**

**Prototype**

```c
unsigned int db_time_change(
    SQLCA * sqlca);
```
Description

sqlca  A pointer to a SQLCA structure. For information, see “The SQL Communication Area (SQLCA)” on page 555.

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.

Returns TRUE if successful, and FALSE otherwise.

fill_s_sqlda function

Prototype

struct sqlda * fill_s_sqlda(
struct sqlda * sqlda,
unsigned int maxlen );

Description

The same as fill_sqlda, except that it changes all the data types in sqlda 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 (sqllen) are modified appropriately. Returns sqlda if successful and returns the null pointer if there is not enough memory available.

The SQLDA should be freed using the free_filled_sqlda function.

fill_sqlda function

Prototype

struct sqlda * fill_sqlda( struct sqlda * sqlda );

Description

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. Returns sqlda if successful and returns the null pointer if there is not enough memory available.

The SQLDA should be freed using the free_filled_sqlda function.
free_filled_sqlda function

Prototype

void free_filled_sqlda( struct sqlda * sqlda );

Description

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 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.

free_sqlda function

Prototype

void free_sqlda( struct sqlda * sqlda );

Description

Free space allocated to this sqlda and free the indicator variable space, as allocated in fill_sqlda. Do not free the memory referenced by each sqldata pointer.

free_sqlda_noind function

Prototype

void free_sqlda_noind( struct sqlda * sqlda );

Description

Free space allocated to this sqlda. Do not free the memory referenced by each sqldata pointer. The indicator variable pointers are ignored.

sql_needs_quotes function

Prototype

unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
Returns a TRUE or FALSE value that indicates whether the string requires double quotes around it when it is used as a SQL identifier. This function formulates a request to the database server to determine if quotes are needed. Relevant information is stored in the sqrcode field.

There are three cases of return value/code combinations:

- **return = FALSE, sqrcode = 0**  The string does not need quotes.
- **return = TRUE**  The sqrcode is always SQLE_WARNING, and the string requires quotes.
- **return = FALSE**  If sqrcode is something other than SQLE_WARNING, the test is inconclusive.

**sqlda_storage function**

Prototype

```c
unsigned int sqlda_storage( struct sqlda * sqlda, int varno );
```

Description

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].

**sqlda_string_length function**

Prototype

```c
unsigned int sqlda_string_length( struct sqlda * sqlda, int varno );
```

Description

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).

**sqlerror_message function**

Prototype

```c
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

Description

Return 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.
Embedded SQL statement summary

EXEC SQL
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** allocate memory for a descriptor. See “ALLOCATE DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference].
- **CLOSE** close a cursor. See “CLOSE statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].
- **CONNECT** connect to the database. See “CONNECT statement [ESQL] [Interactive SQL]” [SQL Anywhere Server - SQL Reference].
- **DEALLOCATE DESCRIPTOR** reclaim memory for a descriptor. See “DEALLOCATE DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference].
- **Declaration section** declare host variables for database communication. See “Declaration section [ESQL]” [SQL Anywhere Server - SQL Reference].
- **DECLARE CURSOR** declare a cursor. See “DECLARE CURSOR statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].
- **DELETE (positioned)** delete the row at the current position in a cursor. See “DELETE (positioned) statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].
- **DESCRIBE** describe the host variables for a particular SQL statement. See “DESCRIBE statement [ESQL]” [SQL Anywhere Server - SQL Reference].
- **DISCONNECT** disconnect from database server. See “DISCONNECT statement [ESQL] [Interactive SQL]” [SQL Anywhere Server - SQL Reference].
- **DROP STATEMENT** free resources used by a prepared statement. See “DROP STATEMENT statement [ESQL]” [SQL Anywhere Server - SQL Reference].
- **EXECUTE** execute a particular SQL statement. See “EXECUTE statement [ESQL]” [SQL Anywhere Server - SQL Reference].
EXPLAIN explain the optimization strategy for a particular cursor. See “EXPLAIN statement [ESQL]” [SQL Anywhere Server - SQL Reference].

FETCH fetch a row from a cursor. See “FETCH statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].

GET DATA fetch long values from a cursor. See “GET DATA statement [ESQL]” [SQL Anywhere Server - SQL Reference].

GET DESCRIPTOR retrieve information about a variable in a SQLDA. See “GET DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference].

GET OPTION get the setting for a particular database option. See “GET OPTION statement [ESQL]” [SQL Anywhere Server - SQL Reference].

INCLUDE include a file for SQL preprocessing. See “INCLUDE statement [ESQL]” [SQL Anywhere Server - SQL Reference].

OPEN open a cursor. See “OPEN statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].

PREPARE prepare a particular SQL statement. See “PREPARE statement [ESQL]” [SQL Anywhere Server - SQL Reference].

PUT insert a row into a cursor. See “PUT statement [ESQL]” [SQL Anywhere Server - SQL Reference].

SET CONNECTION change active connection. See “SET CONNECTION statement [Interactive SQL] [ESQL]” [SQL Anywhere Server - SQL Reference].

SET DESCRIPTOR describe the variables in a SQLDA and place data into the SQLDA. See “SET DESCRIPTOR statement [ESQL]” [SQL Anywhere Server - SQL Reference].

SET SQLCA use a SQLCA other than the default global one. See “SET SQLCA statement [ESQL]” [SQL Anywhere Server - SQL Reference].

UPDATE (positioned) update the row at the current location of a cursor. See “UPDATE (positioned) statement [ESQL] [SP]” [SQL Anywhere Server - SQL Reference].

WHENEVER specify actions to occur on errors in SQL statements. See “WHENEVER statement [ESQL]” [SQL Anywhere Server - SQL Reference].
SQL Anywhere C API reference

Contents

Introduction to the SQL Anywhere C API version 1.0 .................................................... 618
sacapidll.h .................................................................................................................. 619
sacapi.h ..................................................................................................................... 621
a_sqlany_bind_param struct ..................................................................................... 639
a_sqlany_bind_param_info struct .............................................................................. 640
a_sqlany_column_info struct ..................................................................................... 641
a_sqlany_data_info struct .......................................................................................... 642
a_sqlany_data_value struct ....................................................................................... 643
SQLAnywhereInterface struct .................................................................................... 644
a_sqlany_data_direction enumeration ....................................................................... 647
a_sqlany_data_type enumeration ............................................................................. 648
a_sqlany_native_type enumeration ........................................................................... 650
sacapi_error_size constant ........................................................................................ 651
sqlany_current_api_version constant ........................................................................ 652
SQL Anywhere C API examples ................................................................................ 653
Introduction to the SQL Anywhere C API version 1.0

The SQL Anywhere C application programming interface (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. For more information about implementation, see sqlany_imp.sqc.

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 main header file (sacapi.h).

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.

Loading the interface library dynamically

The code to dynamically load the DLL is contained in the header file sacapidll.h. Applications must include the sacapidll.h header file and compile in sacapidll.c. You can use sqlany_initialize_interface to dynamically load the DLL and look up the entry points.
**sacapidll.h**

Dynamically loads the SQL Anywhere C API DLL.

**Syntax**

```c
#define function(x) x ## _func x
```

**Remarks**

You must include `sacapidll.h` in your source files and compile in `sacapidll.c`.

**sqlany_initialize_interface function**

Initializes the SQLAnywhereInterface object and loads the DLL dynamically.

**Syntax**

```c
int sqlany_initialize_interface ( SQLAnywhereInterface * api, const char * optional_path_to_dll )
```

**Parameters**

- **api**  The name of the API structure to initialize.
- **optional_path_to_dll**  Optional argument that specifies a path to a DLL API.

**Remarks**

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).

**Returns**

- 1 on successful initialization, and 0 on failure.

**See also**

- “connecting.cpp” on page 653
- “dbcapi_isql.cpp” on page 654
- “fetching_a_result_set.cpp” on page 657
- “fetching_multiple_from_sp.cpp” on page 659
- “preparing_statements.cpp” on page 661
- “send_retrieve_full_blob.cpp” on page 663
- “send_retrieve_part_blob.cpp” on page 665
sqlany_finalize_interface function

Unloads the library and uninitializes the SQLAnywhereInterface structure.

Syntax

```c
void sqlany_finalize_interface( SQLAnywhereInterface * api )
```

Parameters

- `api` The name of the API structure to initialize.

Remarks

Use sqlany_finalize_interface to finalize and free resources associated with the SQL Anywhere C API DLL.

See also

- “connecting.cpp” on page 653
- “dbcapi_isql.cpp” on page 654
- “fetching_a_result_set.cpp” on page 657
- “fetching_multiple_from_sp.cpp” on page 659
- “preparing_statements.cpp” on page 661
- “send_retrieve_full_blob.cpp” on page 663
- “send_retrieve_part_blob.cpp” on page 665
sacapi.h

Loads the SQL Anywhere C API DLL and looks up all the entry points when the instance is initialized.

Remarks
Only one instance of this element is required in your network environment.

Members
All members of sacapi.h file reference, including all inherited members.

- “a_sqlany_bind_param_info struct” on page 640
- “a_sqlany_column_info struct” on page 641
- “a_sqlany_data_info struct” on page 642
- “a_sqlany_data_value struct” on page 643
- “SQLAnywhereInterface struct” on page 644

sqlany_affected_rows function

Returns the number of rows affected by execution of the prepared statement.

Syntax
sacapi_i32 sqlany_affected_rows( a_sqlany_stmt * 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
- sacapi_i32 The number of rows affected, or -1 on failure.

See also
- “sqlany_execute function” on page 625

sqlany_bind_param function

Binds a user-supplied buffer as a parameter to the prepared statement.

Syntax
sacapi_bool sqlany_bind_param( a_sqlany_stmt * stmt, sacapi_u32 index, a_sqlany_bind_param * param )

Parameters
- stmt A statement prepared successfully using sqlany_prepare.
The index of the parameter. The number must be between 0 and sqlany_num_params() - 1.

- **index**
- **param** An a_sqlany_bind_param structure description of the parameter to be bound.

**Returns**

1 when successful or 0 when unsuccessful.

**See also**

- “sqlany_describe_bind_param function” on page 624

---

**sqlany_clear_error function**

Clears the last stored error code.

**Syntax**

```c
void sqlany_clear_error( a_sqlany_connection * conn )
```

**Parameters**

- **conn** A connection object returned from sqlany_new_connection.

**See also**

- “sqlany_new_connection function” on page 634

---

**sqlany_client_version function**

Returns the current client version.

**Syntax**

```c
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.

**Returns**

1 when successful or 0 when unsuccessful.

---

**sqlany_commit function**

Commits the current transaction.

---
Syntax
sacapi_bool sqlany_commit( a_sqlany_connection * conn )

Parameters
- **conn**  The connection object on which the commit operation is to be performed.

Returns
1 when successful or 0 when unsuccessful.

See also
- "sqlany_rollback function" on page 637

### sqlany_connect function

Creates a connection to a SQL Anywhere database server using the specified connection object and connection string.

Syntax
sacapi_bool sqlany_connect( a_sqlany_connection * conn, const char * str )

Parameters
- **conn**  The 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.

See also
- "sqlany_new_connection function" on page 634
- "sqlany_error function" on page 625
- “Connection parameters” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere database connections” [SQL Anywhere Server - Database Administration]

Example
```c
a_sqlany_connection * conn;
conn = sqlany_new_connection();
if( !sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {  
    char reason[SACAPI_ERROR_SIZE];
    sacapi_i32 code;
    code = sqlany_error( conn, reason, sizeof(reason) );
    printf( "Connection failed. Code: %d Reason: %s\n", code, reason );
} else {   
    printf( "Connected successfully!\n" );
    sqlany_disconnect( conn );
}
sqlany_free_connection( conn );
```
sqlany_describe_bind_param function

Describes the bind parameters of a prepared statement.

Syntax

```c
sacapi_bool sqlany_describe_bind_param( a_sqlany_stmt * stmt,
                                       sacapi_u32 index, a_sqlany_bind_param * param )
```

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.
- **param** The `a_sqlany_bind_param` structure that is populated with information.

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.

Returns

1 on success or 0 on failure.

See also

- “sqlany_bind_param function” on page 621
- “sqlany_prepare function” on page 635

sqlany_disconnect function

Disconnects a SQL Anywhere connection. All uncommitted transactions are rolled back.

Syntax

```c
sacapi_bool sqlany_disconnect( a_sqlany_connection * conn )
```

Parameters

- **conn**  A connection object with a connection established using `sqlany_connect`.

Returns

1 on success or 0 on failure.

See also

- “sqlany_connect function” on page 623
- “sqlany_new_connection function” on page 634
sqlany_error function

Returns the last error code and message stored in the connection object.

Syntax

```c
sacapi_i32 sqlany_error( a_sqlany_connection * conn, char * buffer, size_t size )
```

Parameters

- `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 is success.

See also

- “sqlany_connect function” on page 623
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

sqlany_execute function

Executes a prepared statement.

Syntax

```c
sacapi_bool sqlany_execute( a_sqlany_stmt * stmt )
```

Parameters

- `stmt` A statement prepared successfully using `sqlany_prepare`.

Returns

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 635

Example

```c
// This 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( 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 );
}

sqlany_execute_direct function

Executes the SQL statement specified by the string argument.

Syntax

a_sqlany_stmt * sqlany_execute_direct( a_sqlany_connection * conn, const char * 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 ?.

Remarks

Use this function if you want to prepare and execute a statement, or instead of sqlany_prepare followed by
sqlany_execute. Do not use this function to execute a SQL statement with parameters.

Returns

A statement handle when the function executes successfully, NULL when the function executes
unsuccesfully.

See also

- “sqlany_fetch_absolute function” on page 627
- “sqlany_fetch_next function” on page 628
- “sqlany_num_cols function” on page 634
- “sqlany_get_column function” on page 629

Example

stmt = sqlany_execute_direct( conn, "select * from employees" ) ;
if( stmt ) {
    while( sqlany_fetch_next( stmt ) ) {

int i;
for( i = 0; i < sqlany_num_cols( stmt ); i++ ) {
    // Get i'th column data
}
sqlany_free_stmt( stmt );

**sqlany_execute_immediate function**

Executes the specified SQL statement immediately without returning a result set.

**Syntax**

```c
sacapi_bool sqlany_execute_immediate( a_sqlany_connection * conn, const char * sql )
```

**Parameters**

- **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.

**See also**

- “sqlany_error function” on page 625

**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**

```c
sacapi_bool sqlany_fetch_absolute( a_sqlany_stmt * stmt, sacapi_i32 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**

1 if the fetch is successful, 0 when the fetch is unsuccessful.

**See also**

- “sqlany_error function” on page 625
- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_fetch_next function” on page 628
**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**

```
sacapi_bool sqlany_fetch_next( a_sqlany_stmt * stmt )
```

**Parameters**

- `stmt` A statement object that was executed by sqlany_execute or sqlany_execute_direct.

**Returns**

1 if the fetch is successful, 0 when the fetch is unsuccessful.

**See also**

- “sqlany_error function” on page 625
- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_fetch_absolute function” on page 627

**sqlany_fini function**

Frees resources allocated by the API.

**Syntax**

```
void sqlany_fini( void )
```

**See also**

- “sqlany_init function” on page 633

**sqlany_free_connection function**

Frees the resources associated with a connection object.

**Syntax**

```
void sqlany_free_connection( a_sqlany_connection * conn )
```

**Parameters**

- `conn` A connection object created by sqlany_new_connection.

**See also**

- “sqlany_new_connection function” on page 634
sqlany_free_stmt function

Frees resources associated with a prepared statement object.

Syntax

```c
void sqlany_free_stmt( a_sqlany_stmt * stmt )
```

Parameters

- `stmt`  A statement object returned by the successful execution of sqlany_prepare or sqlany_execute_direct.

See also

- “sqlany_prepare function” on page 635
- “sqlany_execute_direct function” on page 626

sqlany_get_bind_param_info function

Retrieves information about the parameters that were bound using sqlany_bind_param.

Syntax

```c
sacapi_bool sqlany_get_bind_param_info( a_sqlany_stmt * stmt, sacapi_u32 index, a_sqlany_bind_param_info * info )
```

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.
- `info`  The 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 function” on page 621
- “sqlany_describe_bind_param function” on page 624
- “sqlany_prepare function” on page 635

sqlany_get_column function

Fills the supplied buffer with the value fetched for the specified column.
Syntax

sacapi_bool sqlany_get_column(a_sqlany_stmt * stmt, sacapi_u32 col_index,
a_sqlany_data_value * value)

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.
- **value**  The a_sqlany_data_value object to be filled with the data fetched for column col_index.

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. Copy the data from the pointers into the buffers.

The 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 a column contains a 2 GB blob, the sqlany_get_column function attempts to allocate enough memory to hold that value. If you do not want to allocate memory, use sqlany_get_data.

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.

See also

- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_fetch_absolute function” on page 627
- “sqlany_fetch_next function” on page 628

sqlany_get_column_info function

Adds column information to the a_sqlany_column_info structure.

Syntax

sacapi_bool sqlany_get_column_info(a_sqlany_stmt * stmt, sacapi_u32 col_index,
a_sqlany_column_info * info)

Parameters

- **stmt**  A statement object created by sqlany_prepare or sqlany_execute_direct.
- **col_index**  The column number between 0 and sqlany_num_cols - 1.
- **info**  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 function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_prepare function” on page 635

sqlany_get_data function

Retrieves the data fetched for the specified column into the supplied buffer memory.

Syntax

```c
sacapi_i32 sqlany_get_data ( a_sqlany_stmt * stmt, sacapi_u32 col_index, size_t offset, void * buffer, size_t size )
```

Parameters

- **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 2 GB.

Returns

The number of bytes successfully copied into the supplied buffer. This number will not exceed 2 GB. 0 indicates no data remains to be copied. A -1 indicates a failure.

See also

- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_fetch_absolute function” on page 627
- “sqlany_fetch_next function” on page 628

sqlany_get_data_info function

Retrieves information about the data that was fetched by the last fetch operation.

Syntax

```c
sacapi_bool sqlany_get_data_info ( a_sqlany_stmt * stmt, sacapi_u32 col_index, a_sqlany_data_info * info )
```
Parameters

- **stmt** A statement object executed by sqlany_execute or sqlany_execute_direct.
- **col_index** The column number between 0 and sqlany_num_cols() - 1.
- **info** A data info buffer to be filled with the metadata about the data fetched.

Returns

1 on success, and 0 on failure. If any of the supplied parameters is invalid, the sqlany_get_data_info function fails.

See also

- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626
- “sqlany_fetch_absolute function” on page 627
- “sqlany_fetch_next function” on page 628

sqlany_get_next_result function

Advances to the next result set in a multiple result set query.

Syntax

```c
sacapi_bool sqlany_get_next_result( a_sqlany_stmt * stmt )
```

Parameters

- **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.

See also

- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626

Example

```c
stmt = sqlany_execute_direct( 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 );
}
```
**sqlany_init function**

Initializes the interface.

**Syntax**

```c
sacapi_bool sqlany_init( const char * app_name, sacapi_u32 api_version, sacapi_u32 * version_available )
```

**Parameters**

- **app_name** A statement object executed by sqlany_execute or sqlany_execute_direct.
- **api_version** The version of the compiled application (use `SQLANY_CURRENT_API_VERSION`).
- **version_available** The maximum supported API version.

**Returns**

1 on success, 0 otherwise

**See also**

- “sqlany_fini function” on page 628

**Example**

```c
SQLAnywhereInterface api;
unsigned int          max_api_ver;

if( !sqlany_initialize_interface( &api, NULL ) ) {
    printf( "Could not initialize the interface!\n" );
    exit( 0 );
}

if( !api.sqlany_init( "MyAPP", SQLANY_CURRENT_API_VERSION, &max_api_ver ) ) {
    printf( "Failed to initialize the interface! Supported version = %d\n", max_api_ver );
    sqlany_finalize_interface( &api );
    return -1;
}
```

---

**sqlany_make_connection function**

Creates a connection object based on a supplied DBLIB SQLCA pointer.

**Syntax**

```c
a_sqlany_connection * sqlany_make_connection( void * arg )
```

**Parameters**

- **arg** A void * pointer to a DBLIB SQLCA object.
sqlany_new_connection function

Creates a connection object.

Syntax

a_sqlany_connection * sqlany_new_connection( void )

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.

Returns

A connection object.

See also

● “sqlany_connect function” on page 623
● “sqlany_disconnect function” on page 624

sqlany_num_cols function

Returns number of columns in the result set.

Syntax

sacapi_i32 sqlany_num_cols( a_sqlany_stmt * stmt )

Parameters

● 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 function” on page 625
● “sqlany_execute_direct function” on page 626
● “sqlany_prepare function” on page 635
sqlany_num_params function

Returns the number of parameters that are expected for a prepared statement.

Syntax

```c
sacapi_i32 sqlany_num_params( a_sqlany_stmt * stmt )
```

Parameters

- `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 function” on page 635

sqlany_num_rows function

Returns the number of rows in the result set.

Syntax

```c
sacapi_i32 sqlany_num_rows( a_sqlany_stmt * stmt )
```

Parameters

- `stmt` A statement object executed by sqlany_execute or sqlany_execute_direct.

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 [database]” [SQL Anywhere Server - Database Administration].

Returns

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.

See also

- “sqlany_execute function” on page 625
- “sqlany_execute_direct function” on page 626

sqlany_prepare function

Prepares the supplied SQL string
Syntax

```
a_sqlany_stmt * sqlany_prepare( a_sqlany_connection * conn, const char * sql )
```

Parameters

- **conn** A connection object with a connection established using sqlany_connect.
- **sql** The SQL statement to be prepared.

Returns

A handle to a SQL Anywhere statement object.

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 625
- “sqlany_free_stmt function” on page 629
- “sqlany_num_params function” on page 635
- “sqlany_describe_bind_param function” on page 624
- “sqlany_bind_param function” on page 621

Example

```c
char * str;
a_sqlany_stmt * stmt;

str = "select * from employees where salary >= ?";
stmt = sqlany_prepare( conn, str );
if( stmt == NULL ) {
    // Failed to prepare statement, call sqlany_error() for more info
}
```

**sqlany_reset function**

Resets a statement to its prepared state condition.

Syntax

```
sacapi_bool sqlany_reset( a_sqlany_stmt * stmt )
```

Parameters

- **stmt** A statement prepared successfully using sqlany_prepare.

Returns

1 on success, 0 on failure.
See also
  ● “sqlany_prepare function” on page 635

sqlany_rollback function

Rolls back the current transaction.

Syntax
   sacapi_bool sqlany_rollback( a_sqlany_connection * conn )

Parameters
   ● conn   The connection object on which the rollback operation is to be performed.

Returns
   1 on success, 0 otherwise.

See also
  ● “sqlany_commit function” on page 622

sqlany_send_param_data function

Sends data as part of a bound parameter.

Syntax
   sacapi_bool sqlany_send_param_data( a_sqlany_stmt * stmt, sacapi_u32 index, char * buffer, size_t size )

Parameters
   ● 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.

See also
  ● “sqlany_prepare function” on page 635
sqlany_sqlstate function

Retrieves the current SQL state.

Syntax

```c
size_t sqlany_sqlstate( a_sqlany_connection * conn, char * buffer, size_t size )
```

Parameters

- **conn**  A connection object returned from sqlany_new_connection.
- **buffer**  A buffer to be filled with the current 5-character SQL state.
- **size**  The buffer size.

Returns

The number of bytes copied into the buffer.

See also

- “sqlany_error function” on page 625
- “SQL Anywhere error messages sorted by SQLSTATE” [Error Messages]
**a_sqlany_bind_param struct**

Binds prepared statement parameters for execution.

**Syntax**

```c
public a_sqlany_bind_param
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>a_sqlany_bind_param</td>
<td>The data direction (input, output, input_output).</td>
</tr>
<tr>
<td>name</td>
<td>a_sqlany_bind_param</td>
<td>The bind parameter name.</td>
</tr>
<tr>
<td>value</td>
<td>a_sqlany_bind_param</td>
<td>The value to be set.</td>
</tr>
</tbody>
</table>

**See also**

- “sqlany_execute function” on page 625

**Examples**

To view examples of the a_sqlany_bind_param struct reference syntax, see:

- “preparing_statements.cpp” on page 661
- “send_retrieve_full_blob.cpp” on page 663
- “send_retrieve_part_blob.cpp” on page 665
a_sqlany_bind_param_info struct

Used to bind prepared statement parameters for execution.

Syntax

typedef struct a_sqlany_bind_param_info
{
    char *                name;
    a_sqlany_data_direction     direction;
    a_sqlany_data_value        input_value;
    a_sqlany_data_value        output_value;
} a_sqlany_bind_param_info;

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>a_sqlany_data_direction</td>
<td>Parameter direction.</td>
</tr>
<tr>
<td>input_value</td>
<td>a_sqlany_data_value</td>
<td>Information about the bound input value.</td>
</tr>
<tr>
<td>name</td>
<td>char *</td>
<td>Pointer to name of the parameter.</td>
</tr>
<tr>
<td>output_value</td>
<td>a_sqlany_data_value</td>
<td>Information about the bound output value.</td>
</tr>
</tbody>
</table>

See also

- “sqlany_execute function” on page 625

Examples

To view examples of the a_sqlany_bind_param_info struct reference syntax, see:

- “preparing_statements.cpp” on page 661
- “send_retrieve_full_blob.cpp” on page 663
- “send_retrieve_part_blob.cpp” on page 665
a_sqlany_column_info struct

Used to return column metadata information.

Syntax

typedef struct a_sqlany_column_info
{
    char *            name;
    a_sqlany_data_type        type;
    a_sqlany_native_type    native_type;
    unsigned short        precision;
    unsigned short        scale;
    size_t             max_size;
    sacapi_bool            nullable;
} a_sqlany_column_info;

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_size</td>
<td>size_t</td>
<td>The maximum size a data value in this column can take.</td>
</tr>
<tr>
<td>name</td>
<td>char *</td>
<td>The name of the column (null-terminated). The string can be referenced as long as the result set object is not freed.</td>
</tr>
<tr>
<td>native_type</td>
<td>a_sqlany_native_type</td>
<td>The native type of the column in the database.</td>
</tr>
<tr>
<td>nullable</td>
<td>sacapi_bool</td>
<td>If a value in the column can be null or not.</td>
</tr>
<tr>
<td>precision</td>
<td>unsigned short</td>
<td>Precision.</td>
</tr>
<tr>
<td>scale</td>
<td>unsigned short</td>
<td>Scale.</td>
</tr>
<tr>
<td>type</td>
<td>a_sqlany_data_type</td>
<td>The column data type.</td>
</tr>
</tbody>
</table>

Remarks

You can use sqlany_get_column_info to populate this structure.

Examples

To view an example of the a_sqlany_column_info structure in use, see:

- “dbcapi_isql.cpp” on page 654
**a_sqlany_data_info struct**

Used to return information about a column value in a result set.

**Syntax**

```c
typedef struct a_sqlany_data_info
{
    a_sqlany_data_type type;
    sacapi_bool is_null;
    size_t data_size;
} a_sqlany_data_info;
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_size</td>
<td>size_t</td>
<td>The total number of bytes available to be fetched. This field is only valid after a successful fetch operation.</td>
</tr>
<tr>
<td>is_null</td>
<td>sacapi_bool</td>
<td>Indicates whether the last fetched data is NULL. This field is only valid after a successful fetch operation.</td>
</tr>
<tr>
<td>type</td>
<td>a_sqlany_data_type</td>
<td>Type of the data in the column.</td>
</tr>
</tbody>
</table>

**Remarks**

You can use sqlany_get_data_info to populate this structure with information about what was last retrieved by a fetch operation.

**See also**

- “sqlany_get_data_info function” on page 631

**Examples**

To view an example of the a_sqlany_data_info structure in use, see:

- “send_retrieve_part_blob.cpp” on page 665
a_sqlany_data_value struct

Used to return a description of the attributes of a data value.

Syntax

typedef struct a_sqlany_data_value
{
    char * buffer;
    size_t buffer_size;
    size_t * length;
    a_sqlany_data_type type;
    sacapi_bool * is_null;
} a_sqlany_data_value;

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>char *</td>
<td>Pointer to user supplied buffer of data.</td>
</tr>
<tr>
<td>buffer_size</td>
<td>size_t</td>
<td>The size of the buffer.</td>
</tr>
<tr>
<td>is_null</td>
<td>sacapi_bool *</td>
<td>Pointer to an indicator whether the last fetched data is NULL.</td>
</tr>
<tr>
<td>length</td>
<td>size_t *</td>
<td>Pointer to the number of valid bytes in the buffer. Must be less than buffer_size.</td>
</tr>
<tr>
<td>type</td>
<td>a_sqlany_data_type</td>
<td>The type of the data.</td>
</tr>
</tbody>
</table>

Examples

To view examples of the a_sqlany_data_value structure in use, see:

- “dbcapi_isql.cpp” on page 654
- “fetching_a_result_set.cpp” on page 657
- “send_retrieve_full_blob.cpp” on page 663
- “send_retrieve_part_blob.cpp” on page 665
- “preparing_statements.cpp” on page 661
SQLAnywhereInterface struct

The SQL Anywhere C API interface structure.

Syntax

typedef struct SQLAnywhereInterface {
  /** DLL handle. */
  void * dll_handle;

  /** Flag to know if initialized or not. */
  int initialized;

  /** Pointer to ::sqlany_init() function. */
  function( sqlany_init );

  /** Pointer to ::sqlany_fini() function. */
  function( sqlany_fini );

  /** Pointer to ::sqlany_new_connection() function. */
  function( sqlany_new_connection );

  /** Pointer to ::sqlany_free_connection() function. */
  function( sqlany_free_connection );

  /** Pointer to ::sqlany_make_connection() function. */
  function( sqlany_make_connection );

  /** Pointer to ::sqlany_connect() function. */
  function( sqlany_connect );

  /** Pointer to ::sqlany_disconnect() function. */
  function( sqlany_disconnect );

  /** Pointer to ::sqlany_execute_immediate() function. */
  function( sqlany_execute_immediate );

  /** Pointer to ::sqlany_prepare() function. */
  function( sqlany_prepare );

  /** Pointer to ::sqlany_free_stmt() function. */
  function( sqlany_free_stmt );

  /** Pointer to ::sqlany_num_params() function. */
} SQLAnywhereInterface;
function( sqlany_num_params );

/** Pointer to ::sqlany_describe_bind_param() function. */
function( sqlany_describe_bind_param );

/** Pointer to ::sqlany_bind_param() function. */
function( sqlany_bind_param );

/** Pointer to ::sqlany_send_param_data() function. */
function( sqlany_send_param_data );

/** Pointer to ::sqlany_reset() function. */
function( sqlany_reset );

/** Pointer to ::sqlany_get_bind_param_info() function. */
function( sqlany_get_bind_param_info );

/** Pointer to ::sqlany_execute() function. */
function( sqlany_execute );

/** Pointer to ::sqlany_execute_direct() function. */
function( sqlany_execute_direct );

/** Pointer to ::sqlany_fetch_absolute() function. */
function( sqlany_fetch_absolute );

/** Pointer to ::sqlany_fetch_next() function. */
function( sqlany_fetch_next );

/** Pointer to ::sqlany_get_next_result() function. */
function( sqlany_get_next_result );

/** Pointer to ::sqlany_affected_rows() function. */
function( sqlany_affected_rows );

/** Pointer to ::sqlany_num_cols() function. */
function( sqlany_num_cols );

/** Pointer to ::sqlany_num_rows() function. */
function( sqlany_num_rows );

/** Pointer to ::sqlany_get_column() function. */
function( sqlany_get_column );
/** Pointer to ::sqlany_get_data() function. */
function( sqlany_get_data );
/** Pointer to ::sqlany_get_data_info() function. */
function( sqlany_get_data_info );
/** Pointer to ::sqlany_get_column_info() function. */
function( sqlany_get_column_info );
/** Pointer to ::sqlany_commit() function. */
function( sqlany_commit );
/** Pointer to ::sqlany_rollback() function. */
function( sqlany_rollback );
/** Pointer to ::sqlany_client_version() function. */
function( sqlany_client_version );
/** Pointer to ::sqlany_error() function. */
function( sqlany_error );
/** Pointer to ::sqlany_sqlstate() function. */
function( sqlany_sqlstate );
/** Pointer to ::sqlany_clear_error() function. */
function( sqlany_clear_error );
}

\} SQLAnywhereInterface;

Remarks
Only one instance of this structure is required in your application environment. This structure is initialized by the sqlany_initialize_interface function. It attempts to load the SQL Anywhere C API Dynamic Link Library or shared object dynamically and looks up all the entry points of the DLL. The fields in the SQLAnywhereInterface structure will be populated to point to the corresponding functions in the DLL.

See also
- “sqlany_initialize_interface function” on page 619
a_sqlany_data_direction enumeration

A data direction enumeration.

Syntax

typedef enum a_sqlany_data_direction
{
   DD_INVALID    = 0x0,
   DD_INPUT      = 0x1,
   DD_OUTPUT     = 0x2,
   DD_INPUT_OUTPUT = 0x3
} a_sqlany_data_direction;

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD_INVALID</td>
<td>a_sqlany_data_direction</td>
<td>Invalid data direction.</td>
</tr>
<tr>
<td>DD_INPUT</td>
<td>a_sqlany_data_direction</td>
<td>Input only host variables.</td>
</tr>
<tr>
<td>DD_OUTPUT</td>
<td>a_sqlany_data_direction</td>
<td>Output only host variables.</td>
</tr>
<tr>
<td>DD_INPUT_OUTPUT</td>
<td>a_sqlany_data_direction</td>
<td>Input and Output host variables.</td>
</tr>
</tbody>
</table>

a_sqlany_data_type enumeration

Specifies the data type that is being passed in or retrieved.

Syntax

typedef enum a_sqlany_data_type

}{
   A_INVALID_TYPE,
   A_BINARY,
   A_STRING,
   A_DOUBLE,
   A_VAL64,
   A_UVAL64,
   A_VAL32,
   A_UVAL32,
   A_UVAL16,
   A_VAL8,
   A_UVAL8
} a_sqlany_data_type

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_INVALID_TYPE</td>
<td>a_sqlany_data_type</td>
<td>Invalid data type.</td>
</tr>
<tr>
<td>A_BINARY</td>
<td>a_sqlany_data_type</td>
<td>Binary data. Binary data is treated as-is and no character set conversion is performed.</td>
</tr>
<tr>
<td>A_STRING</td>
<td>a_sqlany_data_type</td>
<td>String data. Data where character set conversion is performed.</td>
</tr>
<tr>
<td>A_DOUBLE</td>
<td>a_sqlany_data_type</td>
<td>Double data. Includes float values.</td>
</tr>
<tr>
<td>A_VAL64</td>
<td>a_sqlany_data_type</td>
<td>64-bit integer.</td>
</tr>
<tr>
<td>A_UVAL64</td>
<td>a_sqlany_data_type</td>
<td>64-bit unsigned integer.</td>
</tr>
<tr>
<td>A_VAL32</td>
<td>a_sqlany_data_type</td>
<td>32-bit integer.</td>
</tr>
<tr>
<td>A_UVAL32</td>
<td>a_sqlany_data_type</td>
<td>32-bit unsigned integer.</td>
</tr>
<tr>
<td>A_VAL16</td>
<td>a_sqlany_data_type</td>
<td>16-bit integer.</td>
</tr>
<tr>
<td>A_UVAL16</td>
<td>a_sqlany_data_type</td>
<td>16-bit unsigned integer.</td>
</tr>
<tr>
<td>A_VAL8</td>
<td>a_sqlany_data_type</td>
<td>8-bit integer.</td>
</tr>
</tbody>
</table>
### a_sqlany_data_type enumeration

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_UVAL8</td>
<td>a_sqlany_data_type</td>
<td>8-bit unsigned integer.</td>
</tr>
</tbody>
</table>
a_sqlany_native_type enumeration

An enumeration of the Embedded SQL (ESQL) data types.

Syntax

typedef enum a_sqlany_native_type
{
    DT_NOTYPE = 0,
    DT_DATE = 384,
    DT_TIME = 388,
    DT_TIMESTAMP = 392,
    DT_VARCHAR = 448,
    DT_FIXCHAR = 452,
    DT_LONGVARCHAR = 456,
    DT_STRING = 460,
    DT_DOUBLE = 480,
    DT_FLOAT = 482,
    DT_DECIMAL = 484,
    DT_INT = 496,
    DT_SMALLINT = 500,
    DT_BINARY = 524,
    DT_LONGBINARY = 528,
    DT_TINYINT = 604,
    DT_BIGINT = 608,
    DT_UNSINT = 612,
    DT_UNSSMALLINT = 616,
    DT_UNSBIGINT = 620,
    DT_BIT = 624,
    DT_LONGNVARCHAR = 640
} a_sqlany_native_type;

See also

- “a_sqlany_column_info struct” on page 641
- “sqlany_get_column_info function” on page 630
- “Embedded SQL data types” on page 542
sacapi_error_size constant

Returns the error buffer size.

Synopsis

#define SACAPI_ERROR_SIZE
sqlany_current_api_version constant

Indicates the current API level.

Synopsis

#define SQLANY_CURRENT_API_VERSION
This is an example of how to create a connection object and connect with it to SQL Anywhere.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sacapidll.h"

int main( int argc, char * argv[] )
{
    SQLAnywhereInterface api;
    a_sqlany_connection *conn;
    unsigned int max_api_ver;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!\n" );
        exit( 0 );
    }

    if( !api.sqlany_init( "MyAPP", SQLANY_CURRENT_API_VERSION,
       &max_api_ver ) ) {
        printf( "Failed to initialize the interface! Supported version = %d
\n", max_api_ver );
        sqlany_finalize_interface( &api );
        return -1;
    }

    /* A connection object needs to be created first */
    conn = api.sqlany_new_connection();

    if( !api.sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {
        /* failed to connect */
        char buffer[SACAPI_ERROR_SIZE];
        int rc;
        rc = api.sqlany_error( conn, buffer, sizeof(buffer) );
        printf( "Failed to connect: error code=%d error message=%s\n",
            rc, buffer );
    } else {
        printf( "Connected successfully!\n" );
        api.sqlany_disconnect( conn );
    }

    /* Must free the connection object or there will be a memory leak */
    api.sqlany_free_connection( conn );
    api.sqlany_fini();
}
```
This example shows how to write an ISQL application using dbcapi.

```c
// *********************************************************************
// Copyright 1994-2008 iAnywhere Solutions, Inc. All rights reserved.
// This sample code is provided AS IS, without warranty or liability
// of any kind.
//
// You may use, reproduce, modify and distribute this sample code
// without limitation, on the condition that you retain the foregoing
// copyright notice and disclaimer as to the original iAnywhere code.
//
// *********************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include "sacapidll.h"

#define max( x, y) ( x >= y ? x : y )

SQLAnywhereInterface api;
a_sqlany_connection *conn;

void print_blob( char * buffer, size_t length )
{ /* */
    size_t I;
    if( length == 0 ) {
        return;
    }
    printf( "0x" );
    I = 0;
    while( I < length ) {
        printf( "%2X", (unsigned char)buffer[I] );
        I++;
    }
}

void execute( char * query )
{ /* */
    a_sqlany_stmt * stmt;
    int err_code;
    char err_mesg[SACAPI_ERROR_SIZE];
    int I;
    int num_rows;
    int length;
}
```

**dbcapi_isql.cpp**

sqlany_finalize_interface( &api );
return 0;

---

654  Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
stmt = api.sqlany_execute_direct( conn, query );
if( stmt == NULL ) {
    err_code = api.sqlany_error( conn, err_mesg, sizeof(err_mesg) );
    printf( "Failed: [%d] '%s'
", err_code, err_mesg );
    return;
}
if( api.sqlany_error( conn, NULL, 0 ) > 0 ) {
    err_code = api.sqlany_error( conn, err_mesg, sizeof(err_mesg) );
    printf( "Warning: [%d] '%s'
", err_code, err_mesg );
}
if( api.sqlany_num_cols( stmt ) == 0 ) {
    printf( "Executed successfully.\n" );
    if( api.sqlany_affected_rows( stmt ) > 0 ) {
        printf( "%d affected rows.\n",
            api.sqlany_affected_rows( stmt ) );
    }
    api.sqlany_free_stmt( stmt );
    return;
}

// first output column header
length = 0;
for( I = 0; I < api.sqlany_num_cols( stmt ); I++ ) {
    a_sqlany_column_info    column_info;
    if( I > 0 ) {
        printf("," );
        length += 1;
    }
    api.sqlany_get_column_info( stmt, I, &column_info );
    printf( "%s", column_info.name );
    length += (int)strlen( column_info.name );
} printf( "\n" );
for( I = 0; I < length; I++ ) {
    printf( "-" );
} printf( "\n" );
num_rows = 0;
while( api.sqlany_fetch_next( stmt ) ) {
    num_rows++;
    for( I = 0; I < api.sqlany_num_cols( stmt ); I++ ) {
        a_sqlany_data_value dvalue;
        api.sqlany_get_column( stmt, I, &dvalue );
        if( I > 0 ) {
            printf("," );
        }
        if( *(dvalue.is_null) ) {
            printf( "(NULL)" );
            continue;
        }
        switch( dvalue.type ) {
            case A_BINARY:
                print_blob( dvalue.buffer, *(dvalue.length) );
                break;
            case A_STRING:
                printf( "'%.s\n", *(dvalue.length), (char *)dvalue.buffer,
                    *(dvalue.length) );
                break;
            case A_VAL64:
                printf( "%lld\n", *(long long*)dvalue.buffer );
                break;
            case A_UVAL64:
                printf( "%lld\n", *(long long*)dvalue.buffer );
                break;
            case A_STRING:
                printf( "'%.s\n", *(dvalue.length), (char *)dvalue.buffer,
                    *(dvalue.length) );
                break;
        }
    }
}
printf( "%lld", *(unsigned long long*)dvalue.buffer);
break;
case A_VAL32:
    printf( "%d", *(int*)dvalue.buffer );
    break;
case A_UVAL32:
    printf( "%u", *(unsigned int*)dvalue.buffer );
    break;
case A_VAL16:
    printf( "%d", *(short*)dvalue.buffer );
    break;
case A_UVAL16:
    printf( "%u", *(unsigned short*)dvalue.buffer );
    break;
case A_VAL8:
    printf( "%d", *(char*)dvalue.buffer );
    break;
case A_UVAL8:
    printf( "%u", *(unsigned char*)dvalue.buffer );
    break;
case A_DOUBLE:
    printf( "%f", *(double*)dvalue.buffer );
    break;
}
printf( "\n" );
}
for( I = 0; I < length; I++ ) {
    printf( "-" );
}
printf( "\n" );
printf( "%d rows returned\n", num_rows );
if( api.sqlany_error( conn, NULL, 0 ) != 100 ) {
    char buffer[256];
    int code = api.sqlany_error( conn, buffer, sizeof(buffer) );
    printf( "Failed: [%d] '%s'\n", code, buffer );
}
printf( "\n" );
fflush( stdout );
api.sqlany_free_stmt( stmt );

int main( int argc, char * argv[] )
/***********************************/
{
    unsigned int max_api_ver;
    char buffer[256];
    int len;
    int ch;
    if( argc < 1 ) {
        printf( "Usage: %s -c <connection_string>\n", argv[0] );
        exit( 0 );
    }
    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Failed to initialize the interface!\n" );
        exit( 0 );
    }
    if( !api.sqlany_init( "isql", SQLANY_CURRENT_API_VERSION, &max_api_ver ) )
    {
        printf( "Failed to initialize the interface! Supported version = %d
" );
    }
This example shows how to fetch data from a result set.

```c
// fetching_a_result_set.cpp
// This example shows how to fetch data from a result set.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
`
#include "sacapidll.h"

int main()
{
    SQLAnywhereInterface api;
    a_sqlany_connection * conn;
    a_sqlany_stmt       * stmt;
    unsigned int         max_api_ver;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!\n" );
        exit( 0 );
    }

    if( !api.sqlany_init( "MyAPP", SQLANY_CURRENT_API_VERSION,
                         &max_api_ver ) ) {
        printf( "Failed to initialize the interface! Supported version=%d\n",
                 max_api_ver );
        sqlany_finalize_interface( &api );
        return -1;
    }

    /* A connection object needs to be created first */
    conn = api.sqlany_new_connection();

    if( !api.sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {
        api.sqlany_free_connection( conn );
        api.sqlany_fini();
        sqlany_finalize_interface( &api );
        exit( -1 );
    }

    printf( "Connected successfully!\n" );

    if( (stmt = api.sqlany_execute_direct( conn, "select * from systable") ) != NULL ) {
        num_rows = 0;
        a_sqlany_data_value value;

        while( api.sqlany_fetch_next( stmt ) ) {
            num_rows++;
            printf( "\nRow [%d] data .......
", num_rows );
            for( int I = 0; I < api.sqlany_num_cols( stmt ); I++ ) {
                int rc = api.sqlany_get_column( stmt, I, &value );
                if( rc < 0 ) {
                    printf( "Truncation of column %d\n", I );
                }
                if( *(value.is_null) ) {
                    printf( "Received a NULL value\n" );
                    continue;
                }
                switch( value.type ) {
                    case A_BINARY:
                        printf( "Binary value of length %d.\n",
                                *(value.length) );
                        break;
                    case A_STRING:
                        printf( "String value [%.*s] of length %d.\n",
                                *(value.length), (char *)value.buffer,
                                *(value.length) );
                        break;
                    default:
                }
            }
        }
    }
}
This example shows how to fetch multiple result sets from a stored procedure.

```c
fetching_multiple_from_sp.cpp

This example shows how to fetch multiple result sets from a stored procedure.

// *********************************************************************
// Copyright 1994-2008 iAnywhere Solutions, Inc. All rights reserved.
// This sample code is provided AS IS, without warranty or liability
// of any kind.
```
```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sacapidll.h"

int main()
{
    SQLAnywhereInterface api;
    a_sqlany_connection * conn;
    a_sqlany_stmt * stmt;
    unsigned int max_api_ver;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!\n" );
        exit( 0 );
    }

    if( !api.sqlany_init( "MyAPP", SQLANY_CURRENT_API_VERSION,
                          &max_api_ver ) ) {
        printf( "Failed to initialize the interface! Supported version=%d\n",
                max_api_ver);
        sqlany_finalize_interface( &api);
        return -1;
    }

    /* A connection object needs to be created first */
    conn = api.sqlany_new_connection();

    if( !api.sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {
        api.sqlany_free_connection( conn );
        api.sqlany_fini();
        sqlany_finalize_interface( &api );
        exit( -1 );
    }

    printf( "Connected successfully!\n" );

    api.sqlany_execute_immediate( conn, "drop procedure myproc" );
    api.sqlany_execute_immediate( conn,
                                  "create procedure myproc( ) \n"
                                 "begin \n"
                                 "    select 1, 2; \n"
                                 "    select 3, 4, 5;\n"
                                 "end \n" );

    if( (stmt = api.sqlany_execute_direct( conn, "call myproc()" )) != NULL )
    {
        do {
            /* fetch one row at a time */
            while( api.sqlany_fetch_next( stmt ) ) {
                /* sqlany_num_cols() will be updated everytime the result set shape changes */
                for( int I = 0; I < api.sqlany_num_cols( stmt ); I++ ) {
                    /* process data here ... */
                }
            }
        }
    }
}  ```
/ Check to see if there are other result sets */
} while( api.sqlany_get_next_result( stmt ) );

/* Must free the result set object when done with it */
api.sqlany_free_stmt( stmt );
} api.sqlany_disconnect( conn );

/* Must free the connection object or there will be a memory leak */
api.sqlany_free_connection( conn );
api.sqlany_fini();
sqlany_finalize_interface( &api );
return 0;
}

preparing_statements.cpp

This example shows how to prepare and execute a statement.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sacapidll.h"
#include <assert.h>

int main( )
{
    SQLAnywhereInterface  api;
    a_sqlany_connection * conn;
    a_sqlany_stmt       * stmt;
    unsigned int          max_api_ver;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!\n" );
        exit( 0 );
    }

    if( !api.sqlany_init( "MyAPP", SQLANY_CURRENT_API_VERSION,
        &max_api_ver ) ) { 
        printf( "Failed to initialize the interface! Supported version=%d\n", max_api_ver );
        sqlany_finalize_interface( &api );
        return -1;
    }

    /* A connection object needs to be created first */
    conn = api.sqlany_new_connection();
if( !api.sqlany_connect( conn, "pktdump=c:\temp\pktdump;uid=dba;pwd=sql" ) ) {
    api.sqlany_free_connection( conn );
    api.sqlany_fini();
    sqlany_finalize_interface( &api );
    exit( -1 );
}

printf( "Connected successfully!\n" );

api.sqlany_execute_immediate( conn, "drop procedure myproc" );
api.sqlany_execute_immediate( conn, "create procedure myproc ( IN prefix char(10),
                          INOUT buffer varchar(256),
                          OUT str_len int,
                          IN suffix char(10) )
begin                                       
    set buffer = prefix || buffer || suffix;
    select length( buffer ) into str_len;   
end                                         
" );
stmt = api.sqlany_prepare( conn, "call myproc( ?, ?, ?, ? )" );
if( stmt ) {
    a_sqlany_bind_param param;
    char buffer[256] = "-some_string-";
    int str_len;
    size_t buffer_size = strlen(buffer);
    size_t prefix_length = 6;
    size_t suffix_length = 6;

    assert( api.sqlany_describe_bind_param( stmt, 0, &param ) );
    param.value.buffer = "PREFIX";
    param.value.length = &prefix_length;
    assert( api.sqlany_bind_param( stmt, 0, &param ) );
    assert( api.sqlany_describe_bind_param( stmt, 1, &param ) );
    param.value.buffer = buffer;
    param.value.length = &buffer_size;
    //params[1].value.type      = A_STRING;       // already set by
    sqlany_describe_bind_param(); //params[1].direction       = INPUT_OUTPUT; // already set by
    sqlany_describe_bind_param(); // already set by
    param.value.buffer_size = sizeof(buffer); // IMPORTANT: this
    field must be set for
    // OUTPUT and
    INPUT_OUTPUT parameters so that
    // the library knows
    how much data can be written
    // into the buffer
    assert( api.sqlany_bind_param( stmt, 1, &param ) );

    assert( api.sqlany_describe_bind_param( stmt, 2, &param ) );
    param.value.buffer = (char *)&str_len;
    param.value.is_null = NULL; // use NULL if not
    interested in nullability
    //param.value.type      = A_VAL32; // already set by
    sqlany_describe_bind_param(); //param.direction       = OUTPUT_ONLY; // already set by
    sqlany_describe_bind_param(); // for non string
    or binary buffers, buffer_size is not needed
    assert( api.sqlany_bind_param( stmt, 2, &param ) );
}
assert( api.sqlany_describe_bind_param( stmt, 3, &param ) );
param.value.buffer      = "SUFFIX";
param.value.length      = &suffix_length;
//params.value.type     = A_STRING;               // already set by
sqlany_describe_bind_param()
assert( api.sqlany_bind_param( stmt, 3, &param ) );

/* We are not expecting a result set so the result set parameter could be NULL */
if( api.sqlany_execute( stmt ) ) {
    printf( "Complete string is %s and is %d chars long \n", buffer, str_len );
    assert( str_len == (6+13+6) );
    buffer_size = str_len;
    api.sqlany_execute( stmt );
    printf( "Complete string is %s and is %d chars long \n", buffer, str_len );
    assert( str_len == 6+(6+13+6)+6 );
} else {
    char buffer[SACAPI_ERROR_SIZE];
    int  rc;
    rc = api.sqlany_error( conn, buffer, sizeof(buffer));
    printf( "Failed to execute! [%d] %s\n", rc, buffer );
}

/* Free the statement object or there will be a memory leak */
api.sqlany_free_stmt( stmt );
}

api.sqlany_disconnect( conn );
/* Must free the connection object or there will be a memory leak */
api.sqlany_free_connection( conn );
api.sqlany_fini();
sqlany_finalize_interface( &api );
return 0;
}

send_retrieve_full_blob.cpp

This example shows how to insert and retrieve a blob in one chunk.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "sacapidll.h"
int main()
{
    SQLAnywhereInterface api;
    a_sqlany_connection *conn;
    a_sqlany_stmt       *stmt;
    unsigned int         I;
    unsigned char       *data;
    size_t               size = 1024*1024; // 1MB blob
    int                  code;
    a_sqlany_data_value  value;
    int                  num_cols;
    unsigned int         max_api_ver;
    a_sqlany_bind_param  param;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!\n" );
        exit( 0 );
    }

    assert( api.sqlany_init( "my_php_app", SQLANY_CURRENT_API_VERSION, &max_api_ver ) );
    conn = api.sqlany_new_connection();

    if( !api.sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {
        char buffer[SACAPI_ERROR_SIZE];
        code = api.sqlany_error( conn, buffer, sizeof(buffer) );
        printf( "Could not connection[%d]:%s\n", code, buffer );
        goto clean;
    }

    printf( "Connected successfully!\n" );

    api.sqlany_execute_immediate( conn, "drop table my_blob_table" );
    assert( api.sqlany_execute_immediate( conn, "create table my_blob_table
        (size integer, data long binary)" ) != 0);

    stmt = api.sqlany_prepare( conn, "insert into my_blob_table( size, data )
        values( ?, ?)" );
    assert( stmt != NULL );

    data = (unsigned char *)malloc( size );
    // initialize the buffer
    for( I = 0; I < size; I++ ) {
        data[I] = I % 256;
    }

    // initialize the parameters
    api.sqlany_describe_bind_param( stmt, 0, &param );
    param.value.buffer = (char *)&size;
    param.value.type   = A_VAL32;               // This needs to be set as the
    // server does not
    // know what data will be inserting.
    api.sqlany_bind_param( stmt, 0, &param );

    api.sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = (char *)data;
    param.value.length = &size;
    param.value.type   = A_BINARY;              // This needs to be set for
    // the same reason as above.
    api.sqlany_bind_param( stmt, 1, &param );

    assert( api.sqlany_execute( stmt ) );
}

This example shows how to insert a blob in chunks and retrieve it in chunks too.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "sacapidll.h"

send_retrieve_part_blob.cpp

api.sqlany_free_stmt( stmt );
api.sqlany_commit( conn );
stmt = api.sqlany_execute_direct( conn, "select * from my_blob_table" );
assert( stmt != NULL );
assert( api.sqlany_fetch_next( stmt ) == 1 );
num_cols = api.sqlany_num_cols( stmt );
assert( num_cols == 2 );
api.sqlany_get_column( stmt, 0, &value );
assert( *((int*)value.buffer) == size );
assert( value.type == A_VAL32 );
api.sqlany_get_column( stmt, 1, &value );
assert( value.type == A_BINARY );
assert( *(value.length) == size );
for( I = 0; I < (*value.length); I++ ) {
    assert( (unsigned char)(value.buffer[i]) == data[i]);
}
assert( api.sqlany_fetch_next( stmt ) == 0 );
api.sqlany_free_stmt( stmt );
api.sqlany_disconnect( conn );
clean:  
    api.sqlany_free_connection( conn );
    api.sqlany_fini();
    sqlany_finalize_interface( &api );
    printf("Success!\n");
}
```
int main( )
{
    SQLAnywhereInterface api;
a_sqlany_connection *conn;
a_sqlany_stmt *stmt;
unsigned int I;
unsigned char *data;
unsigned int size = 1024*1024; // 1MB blob
int code;
unsigned char retrieve_buffer[4096];
a_sqlany_data_info dinfo;
size_t total_bytes_read;
unsigned int max_api_ver;

    if( !sqlany_initialize_interface( &api, NULL ) ) {
        printf( "Could not initialize the interface!
" );
        exit( 0 );
    }

    assert( api.sqlany_init( "my_php_app", SQLANY_CURRENT_API_VERSION,
&max_api_ver ) );
    conn = api.sqlany_new_connection();

    if( !api.sqlany_connect( conn, "uid=dba;pwd=sql" ) ) {
        char buffer[SACAPI_ERROR_SIZE];
        code = api.sqlany_error( conn, buffer, sizeof(buffer) );
        printf( "Could not connection[%d]:%s
", code, buffer );
        goto clean;
    }

    printf( "Connected successfully!
" );

    api.sqlany_execute_immediate( conn, "drop table my_blob_table" );
    assert( api.sqlany_execute_immediate( conn, "create table my_blob_table
(size integer, data long binary)" ) != 0);

    // 1. Starting to insert blob operation
    stmt = api.sqlany_prepare( conn, "insert into my_blob_table( size, data)
values( ?, ? )" );
    assert( stmt != NULL );

    // 1.1 We must first bind the parameters
    a_sqlany_bind_param param;

    api.sqlany_describe_bind_param( stmt, 0, &param );
    param.value.buffer = (char *)&size;
    param.value.type = A_VAL32;
    param.value.is_null = NULL;
    param.direction = DD_INPUT;
    api.sqlany_bind_param( stmt, 0, &param );

    api.sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = NULL;
    param.value.type = A_BINARY;
    param.value.is_null = NULL;
    param.direction = DD_INPUT;
    api.sqlany_bind_param( stmt, 1, &param );

    data = (unsigned char *)malloc( size );
for( I = 0; I < size; I++ ) {
    data[i] = I % 256;
}

// 1.2 upload the blob data to the server in chunks
for( I = 0; I < size; I += 4096 ) {
    if( !api.sqlany_send_param_data( stmt, 1, (char *)&data[i], 4096 )) {
        char buffer[SACAPI_ERROR_SIZE];
        code = api.sqlany_error( conn, buffer, sizeof(buffer) );
        printf( "Could not send param[%d]:%s\n", code, buffer );
    }
}

// 1.3 actually do the row insert operation
assert( api.sqlany_execute( stmt ) == 1 );
api.sqlany_commit( conn );
api.sqlany_free_stmt( stmt );

// 2. Now let's retrieve the blob
stmt = api.sqlany_execute_direct( conn, "select * from my_blob_table" );
assert( stmt != NULL );
assert( api.sqlany_fetch_next( stmt ) == 1 );
num_cols = api.sqlany_num_cols( stmt );
assert( num_cols == 2 );
api.sqlany_get_column( stmt, 0, &value );
assert( I == size );
assert( value.type == A_VAL32 );
api.sqlany_get_data_info( stmt, 1, &dinfo );
assert( dinfo.type == A_BINARY );
assert( dinfo.data_size == size );
assert( dinfo.is_null == 0 );

// 2.1 Retrieve data in 4096 byte chunks
total_bytes_read = 0;
while( 1 ) {
    bytes_read = api.sqlany_get_data( stmt, 1, total_bytes_read,
          retrieve_buffer, sizeof(retrieve_buffer) );
    if( bytes_read <= 0 ) {
        break;
    }
    // verify the buffer contents
    for( I = 0; I < (unsigned int)bytes_read; I++ ) {
        assert( retrieve_buffer[i] == data[total_bytes_read+I] );
    }
    total_bytes_read += bytes_read;
} assert( total_bytes_read == size );
free(data );
assert( api.sqlany_fetch_next( stmt ) == 0 );
api.sqlany_free_stmt( stmt );
api.sqlany_disconnect( conn );

clean:
    api.sqlany_free_connection( conn );
    api.sqlany_fini();
    sqlany_finalize_interface( &api );
    printf( "Success!\n" );
}
# SQL Anywhere External Function API

## Contents

- Calling external libraries from procedures ................................................................. 670
- Creating procedures and functions with external calls .............................................. 671
- External function prototypes ...................................................................................... 673
- Using the external function call API methods ............................................................ 681
- Handling data types ................................................................................................... 685
- Unloading external libraries ....................................................................................... 688
Calling external libraries from procedures

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 library call API. Sample external stored procedures, plus the files required to build a DLL containing them, are located in the following folder: `samples-dir\SQLAnywhere\ExternalProcedures`. For information about the location of `samples-dir`, see “Samples directory” [SQL Anywhere Server - Database Administration].

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 API described in this section replaces an older API. The older API is deprecated. Libraries written to the older API, used in versions before version 7.0.x, are still supported, but in any new development, we encourage you to use the new API. Note that the new API 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].
Creating procedures and functions with external calls

This section presents some examples of procedures and functions with external calls.

**DBA authority required**
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 an API described in “External function prototypes” on page 673.

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 PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference]
- “CREATE FUNCTION statement (web services)” [SQL Anywhere Server - SQL Reference]
- “SQL Anywhere external environment support” on page 689
External function prototypes

This section describes the API that you use for functions written in C or C++.

The API 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.

```sql
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:

```c
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 API, your external library must export the following function:

**Syntax**

```c
a_sql_uint32 extfn_use_new_api( );
```

**Returns**

The function returns an unsigned 32-bit integer. The returned value must be the API version number, `EXTFN_API_VERSION`, defined in `extfnapi.h`. A return value of 0 means that the old API is being used.

**Remarks**

If the function is not exported by the library, the database server assumes that the old API is in use. The new API must be used for all Unix platforms and for all 64-bit platforms, including 64-bit Windows.

A typical implementation of this function follows:

```c
a_sql_uint32 extfn_use_new_api( void )
{
    return( EXTFN_API_VERSION );
}
```
extfn_cancel method

To notify the database server that the external library supports cancel processing, your external library must export the following function:

Syntax

```c
void 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:

```c
void extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}
```

See also

- “an_extfn_api structure” on page 674

an_extfn_api structure

Used to communicate with the calling SQL environment.

Syntax

```c
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.

```c
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.

```c
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.

```c
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.

```c
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.

```c
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 676
- “extfn_cancel method” on page 674

**an_extfn_value structure**

Used to access parameter data from the calling SQL environment.

**Syntax**

```c
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 542.

**Remarks**

Suppose that your external function interface was described using the following SQL statement.

```sql
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.

```c
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 674

**an_extfn_result_set_info structure**

Facilitates the return of result sets to the calling SQL environment.

**Syntax**

```c
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 678.
- **column_data_values**  Link to a description of the result set column data. See “an_extfn_result_set_column_data structure” on page 679.

Remarks

The following code fragment shows how to set the properties for objects of this type.

```c
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 678
- “an_extfn_result_set_column_data structure” on page 679

**an_extfn_result_set_column_info structure**

Used to describe a result set.

Syntax

```c
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 542.
- **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.

```c
// 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 677
- “an_extfn_result_set_column_data structure” on page 679

**an_extfn_result_set_column_data structure**

Used to return the data values for columns.

**Syntax**

```c
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.

```c
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 677
- “an_extfn_result_set_column_info structure” on page 678
Using the external function call API methods

get_value callback

```c
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

```c
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:

```sql
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mystring.dll';
```

To call the external function from SQL, we would use a statement like the following:

```sql
call mystring('Hello world!');
```

A sample implementation for the Windows operating system of the mystring function, written in C, follows:

```c
#include <stdio.h>
#include <stdlib.h>
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
#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**

```c
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.

```c
extern "C" __declspec( dllexport )
void extfn_cancel( void *cancel_handle )
{
    *(short *)cancel_handle = 1;
}
```

```c
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 API 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:

```c
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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns 0 when the following is true; else returns 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_value()</td>
<td>● arg_num is invalid; for example, arg_num is greater than the number of arguments for the external function.</td>
</tr>
</tbody>
</table>
| get_piece()   | ● arg_num is invalid; for example, arg_num does not correspond to the argument number used with the previous call to get_value.  
                ● The offset is greater than the total length of the value for the arg_num argument.  
                ● It is called before get_value has been called. |
| set_value()   | ● arg_num is invalid; for example, arg_num is greater than the number of arguments for the external function.  
                ● Argument arg_num is input only.  
                ● The type of value supplied does not match that of argument arg_num. |
## Handling data types

### Data types

The following SQL data types can be passed to an external library:

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>sqldef.h</th>
<th>C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>DT_FIXCHAR</td>
<td>Character data, with a specified length</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>DT_VARCHAR</td>
<td>Character data, with a specified length</td>
</tr>
<tr>
<td>LONG VARCHAR, TEXT</td>
<td>DT_LONGVARCHAR</td>
<td>Character data, with a specified length</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>DT_FIXCHAR</td>
<td>Character data, with a specified length</td>
</tr>
<tr>
<td>XML</td>
<td>DT_LONGVARCHAR</td>
<td>Character data, with a specified length</td>
</tr>
<tr>
<td>NCHAR</td>
<td>DT_NFIXCHAR</td>
<td>UTF-8 character data, with a specified length</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>DT_NVARCHAR</td>
<td>UTF-8 character data, with a specified length</td>
</tr>
<tr>
<td>LONG NVARCHAR, NTEXT</td>
<td>DT_LONGNVARCHAR</td>
<td>UTF-8 character data, with a specified length</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>DT_BINARY</td>
<td>Binary data, 16 bytes long</td>
</tr>
<tr>
<td>BINARY</td>
<td>DT_BINARY</td>
<td>Binary data, with a specified length</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>DT_BINARY</td>
<td>Binary data, with a specified length</td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>DT_LONGBINARY</td>
<td>Binary data, with a specified length</td>
</tr>
<tr>
<td>TINYINT</td>
<td>DT_TINYINT</td>
<td>1-byte integer</td>
</tr>
<tr>
<td>[ UNSIGNED ] SMALLINT</td>
<td>DT_SMALLINT, DT_UNSINT</td>
<td>[ Unsigned ] 2-byte integer</td>
</tr>
<tr>
<td>[ UNSIGNED ] INT</td>
<td>DT_INT, DT_UNSINT</td>
<td>[ Unsigned ] 4-byte integer</td>
</tr>
</tbody>
</table>
### SQL data type

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>sqldef.h</th>
<th>C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ UNSIGNED ] BIGINT</td>
<td>DT_BIGINT, DT_UNSIGNED_BIGINT</td>
<td>[ Unsigned ] 8-byte integer</td>
</tr>
<tr>
<td>REAL, FLOAT(1-24)</td>
<td>DT_FLOAT</td>
<td>Single precision floating point number</td>
</tr>
<tr>
<td>DOUBLE, FLOAT(25-53)</td>
<td>DT_DOUBLE</td>
<td>Double precision floating point number</td>
</tr>
</tbody>
</table>

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 API function. To read IN and INOUT parameters, use the get_value API 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.

```c
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 on data types, see “Using host variables” on page 546.

### UTF-8 types

The UTF-8 data types such as NCHAR, NVARCHAR, LONG NVARCHAR and NTTEXT as 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.

```c
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.

```sql
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

Contents

Overview of external environments ................................................................. 690
The CLR external environment ........................................................................ 694
The ESQL and ODBC external environments .................................................. 697
The Java external environment ...................................................................... 706
The PERL external environment ................................................................... 711
The PHP external environment ...................................................................... 715
Overview of external environments

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:

```sql
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. In the case of 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. In the case of one-per-database, the 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. In most cases, though, 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.

**SQL statements**

The following SQL syntax allows you to set or modify values in the SYS.SYSEXTERNENV table.
ALTER EXTERNAL ENVIRONMENT environment-name
  [ USER user-name ]
  [ LOCATION location-string ]

- **environment-name**  The environment name is an identifier representing the name of the environment in SYS.SYSEXTERNENV. It is one of PERL, PHP, JAVA, CLR, C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

- **user-name**  The user name string identifies a user in the database who 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 user-name would indicate the different user ID to be used. In most cases, this option need not be specified.

- **location-string**  The location string 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.

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 add a comment for an external environment, you can execute:

```
COMMENT ON EXTERNAL ENVIRONMENT environment-name
  IS comment-string
```

To install an 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
```

- **object-name-string**  The object name string is the name by which the installed object is identified within the database.

- **update-mode**  The update mode is either NEW or UPDATE. If the update mode is omitted, then NEW is assumed.

- **file-path**  The file path is the location on the database server computer from where the object is installed.

- **environment-name**  The environment name is one of JAVA, PERL, PHP, CLR, C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

To add a comment for an installed external object, you can execute:

```
COMMENT ON EXTERNAL ENVIRONMENT OBJECT object-name-string
  IS comment-string
```
To remove an installed external object from the database, you would need to use a REMOVE EXTERNAL OBJECT statement:

```
REMOVE EXTERNAL OBJECT object-name-string
```

- **object-name-string**  The object name string is the same string that was specified in the corresponding INSTALL EXTERNAL OBJECT statement.

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
```

- **environment-name**  The environment name is one of JAVA, PERL, PHP, CLR, C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

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
```

- **environment-name**  Environment name is one of JAVA, PERL, PHP, CLR, C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

For more information, see:

- “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 (web services)” [SQL Anywhere Server - SQL Reference]
- “CREATE PROCEDURE statement (web services)” [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 with the exception 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 datatypes:

<table>
<thead>
<tr>
<th>CLR type</th>
<th>Recommended SQL data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>bit</td>
</tr>
<tr>
<td>byte</td>
<td>tinyint</td>
</tr>
<tr>
<td>short</td>
<td>smallint</td>
</tr>
<tr>
<td>ushort</td>
<td>unsigned smallint</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>uint</td>
<td>unsigned int</td>
</tr>
<tr>
<td>long</td>
<td>bigint</td>
</tr>
<tr>
<td>ulong</td>
<td>unsigned bigint</td>
</tr>
<tr>
<td>CLR type</td>
<td>Recommended SQL data type</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>decimal</td>
<td>numeric</td>
</tr>
<tr>
<td>float</td>
<td>real</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DateTime</td>
<td>timestamp</td>
</tr>
<tr>
<td>string</td>
<td>long varchar</td>
</tr>
<tr>
<td>byte[]</td>
<td>long binary</td>
</tr>
</tbody>
</table>

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 `samples-dir\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 `dbextclr11.exe`. Make sure that this file is present in the `install-dir\Bin32` or `install-dir\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 prior to using of the CLR external environment.

Here is an example of a function written in C# that can be run within an external environment.

---

The CLR 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 dbextclr11.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. In the case of 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. In the case of 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 additional information and examples on using the CLR in the database support, refer to the examples located in the samples-dir\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 API to pass information to and return information from the native function. This API is described in “SQL Anywhere External Function API” on page 669.

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 prior to 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.

```c++
#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;
}
```
When compiled into a dynamic link library or shared object, this function can be called from an external environment. An executable image called `dbexternc11` 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 `dbexternc11`. This is one of the advantages of using the external environment. Note that once `dbexternc11` 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 `dbexternc11`.

To call the compiled native function, SimpleCFunction, a wrapper is defined as follows:

```sql
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-
bite 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.

```sql
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.

```c
#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:\sa11 and that Microsoft Visual C++ is installed).

    cl extodbc.cpp /LD /Ic:\sa11\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;
    }
If the above embedded SQL code is 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:\sa11` and that Microsoft Visual C++ is installed).

```
sqlpp extesql.sqc extesql.cpp
cl extesql.cpp /LD /ic:\sa11\sdk\include c:\sa11\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.

```sql
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:\sa11 and that Microsoft Visual C++ is installed).

cl /LD /Tp extcapi.c /Tp c:\sa11\SDK\C\sacapidll.c /Ic:\sa11\SDK\Include c:\sa11\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 dbexternc11. 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 dbexternc11. 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_ESQL64

Another case where preloading dbexternc11 is useful is when you want to debug your external function. You can use the debugger to attach to the running dbexternc11 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, dbexternc11, 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 dbexternc11 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_ESQL64

To return a result set from an external function, the compiled native function must use the native function call API. This API is fully described in “SQL Anywhere External Function API” on page 669. The following are some of the highlights for returning result sets.
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.

```c
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.

```c
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 );
    }
```

*SQL Anywhere external environment support*

704 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
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.

```c
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 );
        extapi->set_value( arg_handle, 0, &retval, 0 );
        return;
    }
}
```

For additional information, see “SQL Anywhere External Function API” on page 669.

For more information on how to make server-side requests and how to return result sets from an external function, refer to the samples in `samples-dir\SQLAnywhere\ExternalEnvironments\ExternC`. 
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 with the exception 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 Virtual Machine environment). It should be noted that there is once 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';
```

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 all connections terminate to the database. However, if you are completely done with Java and you want to make it possible for the database 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
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
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 in excess of 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:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>byte</td>
</tr>
<tr>
<td>C</td>
<td>char</td>
</tr>
<tr>
<td>D</td>
<td>double</td>
</tr>
<tr>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>I</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>long</td>
</tr>
<tr>
<td>L class-name;</td>
<td>an instance of the class class-name. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String</td>
</tr>
<tr>
<td>S</td>
<td>short</td>
</tr>
<tr>
<td>V</td>
<td>void</td>
</tr>
</tbody>
</table>
The Java external environment

<table>
<thead>
<tr>
<th>Field type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Boolean</td>
</tr>
<tr>
<td>[</td>
<td>use one for each dimension of an array</td>
</tr>
</tbody>
</table>

For example,

```java
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).

```sql
CREATE PROCEDURE insertfix()
EXTERNAL NAME 'JDBCExample.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).

```sql
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).

```sql
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
```

For more information about calling Java methods, see “Accessing methods in the Java class” on page 95.

For more information on returning result sets, see “Returning result sets from Java methods” on page 103.

The following Java example takes a string and writes it to the database server messages window:

```java
import java.io.*;
public class Hello
{
    public static void main( String[] args )
    {
        System.out.print( "Hello" );
        for ( int i = 0; i < args.length; i++ )
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1  709
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.

For additional information and examples on using the Java in the database support, refer to “Java support in SQL Anywhere” on page 77.
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 with the exception 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. In the case of Java, there is once 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 “SQL Anywhere Perl DBD::SQLAnywhere DBI module” on page 721.

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:

To install the external environment module (Windows)

- Run the following commands from the $DK\PerlEnv subdirectory of your SQL Anywhere installation:

  perl Makefile.PL
  nmake
  nmake install

To install the external environment module (Unix)

- Run the following commands from the $dk/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.

\[
\text{ALTER EXTERNAL ENVIRONMENT PERL} \\
\quad \text{LOCATION 'perl-path';}
\]

For example:

\[
\text{ALTER EXTERNAL ENVIRONMENT PERL} \\
\quad \text{LOCATION 'c:\PerI\bin\perl.exe';}
\]

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.

\[
\text{INSTALL EXTERNAL OBJECT 'perl-script'} \\
\quad \text{NEW} \\
\quad \text{FROM FILE 'perl-file'} \\
\quad \text{ENVIRONMENT PERL;}
\]

Perl code also can be built and installed from an expression, as follows:

\[
\text{INSTALL EXTERNAL OBJECT 'perl-script'} \\
\quad \text{NEW} \\
\quad \text{FROM VALUE 'perl-statements'} \\
\quad \text{ENVIRONMENT PERL;}
\]

Perl code also can be built and installed from a variable, as follows:

\[
\text{CREATE VARIABLE PerlVariable LONG VARCHAR;} \\
\text{SET PerlVariable = 'perl-statements';} \\
\text{INSTALL EXTERNAL OBJECT 'perl-script'} \\
\quad \text{NEW} \\
\quad \text{FROM VALUE PerlVariable} \\
\quad \text{ENVIRONMENT PERL;}
\]

To remove Perl code from the database, use the REMOVE statement, as follows:

\[
\text{REMOVE EXTERNAL OBJECT 'perl-script'}
\]

To modify existing Perl code, you can use the UPDATE clause of the INSTALL EXTERNAL OBJECT statement, as follows:

\[
\text{INSTALL EXTERNAL OBJECT 'perl-script'} \\
\quad \text{UPDATE} \\
\quad \text{FROM FILE 'perl-file'} \\
\quad \text{ENVIRONMENT PERL}
\]
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 datatypes are mapped to strings when making the Perl call while binary data is mapped to an array of numbers. A simple Perl example follows:

```sql
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:
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 additional information and examples on using the Perl in the database support, refer to the examples located in the `samples-dir\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 with the exception 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. In the case of Java, there is once 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 driver (shipped with SQL Anywhere) must be installed on the database server computer. See “Installing and configuring SQL Anywhere PHP” on page 741.

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.

To 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. The following is a model to use:

```
copy install-dir\Bin32\php-5.2.6_sqlanywhere_extenv11.dll php-dir\ext
```
3. 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.

To 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. The following is a model to use:
3. 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_extenv11` modules are located in the directory indicated by `extension_dir`. 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`, `dblib11.dll`, `dbicu11.dll`, `dbicudt11.dll`, `dblgen11.dll`, and `dbextenv11.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`, `libdblib11_r`, `libdbicu11_r`, `libdbicudt11`, `dblgen11.res`, and `libdbextenv11_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 datatypes 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:

```sql
INSTALL EXTERNAL OBJECT 'SimplePHPExample'
NEW
FROM VALUE '<? 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 ("""). 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:

```sql
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<? 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.

```sql
CREATE TABLE phpTab(c1 int, c2 char(128));

INSTALL EXTERNAL OBJECT 'ServerSidePHPExample'
NEW
FROM VALUE '<? 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
'<?php
  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 additional information and examples on using the PHP in the database support, refer to the examples located in the `samples-dir\SQLAnywhere\ExternalEnvironments\PHP` directory.
SQL Anywhere Perl DBD::SQLAnywhere DBI module

Contents

Introduction to DBD::SQLAnywhere .............................................................................. 722
Installing DBD::SQLAnywhere on Windows .................................................................... 723
Installing DBD::SQLAnywhere on Unix and Mac OS X .................................................... 725
Writing Perl scripts that use DBD::SQLAnywhere ............................................................ 727
Introduction to DBD::SQLite

The DBD::SQLite interface provides access to SQLite databases from scripts written in Perl. DBD::SQLite 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::SQLite, you can access and change the information in SQLite databases from Perl.

The DBD::SQLite driver is thread-safe when using Perl with pthreads.

Requirements

The DBD::SQLite interface requires the following components.

- Perl 5.6.0 or newer. On Windows, ActivePerl 5.6.0 build 616 or later is required.
- DBI 1.34 or newer.
- A C compiler. On Windows, only the Microsoft Visual C++ compiler is supported.

The following sections provide assistance with installing Perl, DBI, and the DBD::SQLite driver software.
Installing DBD::SQLAnywhere on Windows

**To 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 example, a batch file called `vcvars32.bat` is included in the `vc\bin` subdirectory of the Visual Studio 2005 or 2008 installation. Open a new system command prompt and run this batch file before continuing.

**To 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.

<table>
<thead>
<tr>
<th>name</th>
<th>version</th>
<th>abstract</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBI</td>
<td>1.58</td>
<td>Database independent interface for Perl</td>
<td>perl</td>
</tr>
</tbody>
</table>

   If DBI is not installed, you must install it. To do so, enter the following command at the ppm prompt.

   ```
   ppm install dbi
   ```

**To 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.

```bash
copy "samples-dir\demo.db" .
```

For information about the default location of `samples-dir`, see “Samples directory” [SQL Anywhere Server - Database Administration].

```bash
dbeng11 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.

```bash
nmake install
```

The DBD::SQLAnywhere interface is now ready to use.
Installing DBD::SQLAnywhere 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.

To prepare your computer

1. Install ActivePerl 5.6.0 build 616 or later.
2. Install a C compiler.

To install the DBI Perl module on Unix and Mac OS X

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.

To 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:

<table>
<thead>
<tr>
<th>In this shell ...</th>
<th>... use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh, ksh, or bash</td>
<td>. bin/sa_config.sh</td>
</tr>
<tr>
<td>csh or tcsh</td>
<td>source bin/sa_config.csh</td>
</tr>
</tbody>
</table>

2. At a shell prompt, change to the $dk/perl subdirectory of your SQL Anywhere installation.
3. At a command prompt, run the following commands to build DBD::SQLAnywhere.

   perl Makefile.PL
If for any reason you need to start over, you can use the command `make clean` to remove any partially built targets.

4. To test DBD::SQLAnywhere, copy the sample database file to your `sdk/perl` directory and make the tests.

   ```bash
   cp samples-dir/demo.db .
   dbeng11 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.

5. To complete the installation, execute the following command at the same prompt.

   ```bash
   make install
   ```

   The DBD::SQLAnywhere interface is now ready to use.
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 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.

```perl
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.

```perl
#!/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.

```perl
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:ENG=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my %defaults = (
```
SQL Anywhere Perl DBD::SQLAnywhere DBI module

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 $data_src = "DBI:SQLAnywhere:ENG=$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";
}

728

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1


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 “max_cursor_count option [database]” [SQL Anywhere Server - Database Administration], and “max_statement_count option [database]” [SQL Anywhere Server - Database Administration].

### 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 place holders 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.

```perl
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:ENG=$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)
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";
$db_insert($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 = undef;
    }
__END__
```
$sth = $dbh->prepare($ins);
my $row = undef;
foreach $row (@rows) {
    my @values = split(/,/, $row);
    $sth->execute(@values);
}
__END__
SQL Anywhere Python Database support

Contents

Introduction to sqlanydb ................................................................. 732
Installing sqlanydb on Windows ...................................................... 733
Installing sqlanydb on Unix and Mac OS X ...................................... 734
Writing Python scripts that use sqlanydb ........................................ 735
Introduction to sqlanydb

The sqlanydb interface provides access to SQL Anywhere databases from scripts written in Python. 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, visit Python Database API specification v2.0.

The sqlanydb module is thread-safe when using Python with threads.

Requirements

The sqlanydb module requires the following components.

- Python 2.4 or newer (2.5 or newer is recommended).
- 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.

```python
import ctypes
```

If you see an error message, then ctypes is not present. The following is an example.

```python
>>> 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, or can be automatically downloaded and installed with Peak EasyInstall. To download Peak EasyInstall, visit http://peak.telecommunity.com/DevCenter/EasyInstall.

The following sections provide assistance with installing Python and the sqlanydb module.
Installing sqlanydb on Windows

**To prepare your computer**

1. Install Python 2.4 or later.
2. Install the ctypes module if missing.

**To 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 "samples-dir\demo.db" .
dbeng11 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 sqlanydb 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.

To prepare your computer

1. Install Python 2.4 or later.
2. Install the ctypes module if missing.

To 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:

<table>
<thead>
<tr>
<th>In this shell ...</th>
<th>... use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh, ksh, or bash</td>
<td>. bin/sa_config.sh</td>
</tr>
<tr>
<td>csh or tcsh</td>
<td>source bin/sa_config.csh</td>
</tr>
</tbody>
</table>

2. At a shell prompt, change to the sdk/python subdirectory of your SQL Anywhere installation.
3. Enter the following command to install sqlanydb.
   python setup.py install
4. To test sqlanydb, copy the sample database file to your sdk/python directory and run a test.
   cp samples-dir/demo.db .
dbeng11 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.
Writing 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 Python Database API specification v2.0.

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.

```python
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.

```python
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 11 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"

For the complete list of connection parameters, see “Connection parameters and network protocol options” [SQL Anywhere Server - Database Administration].

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.

```python
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 11 Demo" )

# Close the connection
con.close()

**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

# Close the connection for row in rowset:
# 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 place holders 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.
SQL Anywhere PHP API

Contents

Introduction to the SQL Anywhere PHP module ....................................................... 740
Installing and configuring SQL Anywhere PHP ......................................................... 741
Running PHP test scripts in your web pages ............................................................ 746
Writing PHP scripts .................................................................................................... 748
SQL Anywhere PHP API reference ........................................................................... 754
Building the SQL Anywhere PHP module on Unix and Mac OS X ......................... 803
Introduction to the SQL Anywhere PHP module

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 are two modules that provide access to SQL Anywhere databases from PHP. You can use these modules and the PHP language to write standalone scripts and create dynamic web pages that rely on information stored in SQL Anywhere databases.

Prebuilt versions of the PHP modules are provided for Windows, Linux, and Solaris and are installed in the operating-system-specific binaries subdirectories of your SQL Anywhere installation. Source code for the SQLAnywhere PHP module is installed in the `sdk/php` subdirectory of your SQL Anywhere installation.

Additional information can also be found online at The SQL Anywhere PHP Module.

Requirements

Before you can use the SQL Anywhere PHP module, you must also install the following components:

- The PHP 5 binaries for your platform, which can be downloaded from [http://www.php.net](http://www.php.net). SQL Anywhere provides prebuilt PHP modules for PHP versions 5.1.1 to 5.2.6. At the time of writing, PHP version 5.2.6 was the most recent stable release. For Windows platforms, the thread-safe version of PHP must be used with the SQL Anywhere PHP module.
- A web server, if you want to run PHP scripts within a web server. Note that SQL Anywhere can be used as a web server.
  
  You can also use a different web server such as Apache HTTP Server. SQL Anywhere can be run on the same computer as the web server, or on a different computer.
- For Windows, the SQL Anywhere client software `dblib11.dll` and `dbcapi.dll`.
- For Linux/Unix, the SQL Anywhere client software `libdblib11.so` and `libdbcap.so`.
- For Mac OS X, the SQL Anywhere client software `libdblib11.dylib` and `libdbapi.dylib`.

For additional information about installing PHP and the Apache HTTP Server, see Serving Content from SQL Anywhere Databases Using Apache and PHP.

The following sections provide assistance with installing the SQL Anywhere PHP module.
Installing and configuring SQL Anywhere PHP

The following sections describe how to install and configure the SQL Anywhere PHP module.

Choosing which PHP module to use

On Windows, SQL Anywhere includes thread-safe modules for PHP versions 5.1.1 to 5.2.6. A thread-safe version of PHP must be used with the SQL Anywhere PHP module. The module file names for the supported PHP versions follow this pattern:

`php-5.x.y_sqlanywhere.dll`

On Linux and Solaris, SQL Anywhere includes both 64-bit and 32-bit versions of the modules for PHP versions 5.1.1 to 5.2.6. It also includes both threaded and non-threaded modules. If you are using the CGI version of PHP or if you are using Apache 1.x, use the non-threaded module. If you are using Apache 2.x, use the threaded module. The module 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, 5.2.6). For Linux and Solaris, the threaded version of the PHP module has 
_r appended to the file name. Windows versions are implemented as Dynamic Link Libraries and Linux/Solaris versions are implemented as Shared Objects.

Installing the PHP module on Windows

To use the SQL Anywhere PHP module 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 module, so you do not need to load it manually in each script.

To install the PHP module 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 modules provided by SQL Anywhere, try using the most recent module provided. Note that a version 5.2.x SQL Anywhere PHP module 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 module 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 “Configuring the SQL Anywhere PHP module” on page 744.

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.

5. At a command prompt, run the following command to start the SQL Anywhere sample database.

   dbeng11 samples-dir\demo.db

   The command starts a database server using the sample database.

6. 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-5.2.6_sqlanywhere.dll
   Connected successfully

   If the SQL Anywhere PHP driver 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.

7. When you are done, stop the SQL Anywhere database server by clicking Shut Down in the database server messages window.

   For more information, see “Creating PHP test pages” on page 746.

### Installing the PHP module on Linux/Solaris

To use the SQL Anywhere PHP module 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 module, so you do not need to load it manually in each script.

**To install the PHP module 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 module will not work with a version 5.3.x PHP.

For information about which version of the shared object to use, see “Choosing which PHP module to use” on page 741.

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. See “Configuring the SQL Anywhere PHP module” on page 744.

4. Before attempting to use the PHP module, 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:

<table>
<thead>
<tr>
<th>In this shell</th>
<th>... use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh, ksh, or bash</td>
<td>. /bin32/sa_config.sh</td>
</tr>
<tr>
<td>csh or tcsh</td>
<td>source /bin32/sa_config.csh</td>
</tr>
</tbody>
</table>

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.

5. At a command prompt, run the following command to start the SQL Anywhere sample database:

```
dbeng11 samples-dir/demo.db
```

6. At a command prompt, change to the sdk/php/examples subdirectory of your SQL Anywhere installation. Enter the following command:
Building the PHP module on Unix and Mac OS X

To use the SQL Anywhere PHP module on other versions of Unix or Mac OS X, you must build the PHP module from the source code which is installed in the `sdk/php` subdirectory of your SQL Anywhere installation. See “Building the SQL Anywhere PHP module on Unix and Mac OS X” on page 803.

Configuring the SQL Anywhere PHP module

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 module 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 module is loaded. The following entry is used for Windows platforms.

  ```python
  extension=php-5.x.y_sqlanywhere.dll
  ```

  On Linux platforms, use one of the following entries. The second entry is thread safe.

  ```python
  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 module 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 module is loaded.

  ```bash
  # Ensure that the SQL Anywhere PHP module is loaded
  if( !extension_loaded('sqlanywhere') ) {
    # Find out which version of PHP is running
    $version = phpversion();
    $module_name = 'php-'.substr($version, 0, 3).'_sqlanywhere';
    if( strtoupper(substr(PHP_OS, 0, 3) == 'WIN' )) {
      $module_ext = '.dll';
    } else {
      $module_ext = '.so';
    }
  }
  ```
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 module. 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

For more information, see “sasql_set_option” on page 772.
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.

To 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
   <?php phpinfo(); ?>
   ```

   Alternatively, once PHP is properly installed and configured, you can also create a test web page by issuing the following command at a command prompt.

   ```
   php -i > info.html
   ```

   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):

   ```
   dbeng11 samples-dir\demo.db
   ```

4. To test that PHP and your web server are working correctly with SQL Anywhere:
   a. Copy the file `connect.php` from your 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.

To create the query page that uses the SQL Anywhere PHP module

1. Create a file containing the following PHP code in your root web content directory named `sa_test.php`. 
2. Insert the following PHP code into this file:

```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 module.

### To 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:

<table>
<thead>
<tr>
<th>In this shell ...</th>
<th>... use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh, ksh, or bash</td>
<td>. /bin32/sa_config.sh</td>
</tr>
<tr>
<td>csh or tcsh</td>
<td>source /bin32/sa_config.csh</td>
</tr>
</tbody>
</table>

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.

   ```
dbeng11 samples-dir\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:

<table>
<thead>
<tr>
<th>For this test page ...</th>
<th>... use this URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>info.php</td>
<td><a href="http://localhost/info.php">http://localhost/info.php</a></td>
</tr>
</tbody>
</table>

   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 module 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.

Connecting to a database

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
<?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 );
    }
?>
```

The first block of code verifies that the PHP module is loaded. If you added the line to your PHP initialization file to load it automatically, this block of code is unnecessary. If you did not configure PHP to automatically load the SQL Anywhere PHP module at start time, you must add this code to the other sample scripts.

The second block attempts to make a connection. For this code to succeed, the SQL Anywhere sample database must be running.

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
<?php
    # Connect using the default user ID and password
    $conn = sasql_connect( "UID=DBA;PWD=sql" );
    if( ! $conn ) {
        echo "Connection failed\n";
```
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*.
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
<?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
<?php
    # Connect using the default user ID and password
    $conn = sasql_connect("UID=DBA;PWD=sql");
```
if( ! $conn ) {
    die ("Connection failed");
} else {
    # Connected successfully.
}
# Retrieve the data and output HTML
echo "<BR>\n";
$query1 = "SELECT table_id, table_name FROM SYSTAB";
$result = sasql_query( $conn, $query1 );
if( $result ) {
    $num_rows = sasql_num_rows( $result );
    echo "Returned : $num_rows <BR>\n";
    $I = 1;
    while( ($row = sasql_fetch_array( $result ))) {
        echo "$I:  table_id:$row[table_id]" .
            " --- table_name:$row[table_name] <br>\n";
        $query2 = "SELECT table_id, column_name " .
            "FROM SYSTABCOL" .
            "WHERE table_id = '$row[table_id]'";
        echo " $query2 <br>\n";
        echo " Columns: ";
        $result2 = sasql_query( $conn, $query2 );
        if( $result2 ) {
            while(($detailed = sasql_fetch_array($result2))) {
                echo "$detailed[column_name]";
            }
            sasql_free_result( $result2 );
        } else {
            echo "******FAILED********
        }
    }
    echo "<BR>\n";
    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";
    $qname = $_POST["qname"];  
    $qname = str_replace( "\", "\", $qname );
    echo "<form method=post action=webisql.php>\n";
    echo "<br>Query: <input type=text Size=80 name=qname value="$qname">\n";
    echo "<input type=submit>\n";
?>
This design could be extended to handle complex web forms by formulating customized SQL queries based on the values entered by the user.

### Working with BLOBs

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.
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.
The PHP API supports the following functions:

**Connections**
- “sasql_close” on page 756
- “sasql_connect” on page 756
- “sasql_disconnect” on page 758
- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_insert_id” on page 764
- “sasql_message” on page 765
- “sasql_pconnect” on page 767
- “sasql_set_option” on page 772

**Queries**
- “sasql_affected_rows” on page 755
- “sasql_next_result” on page 766
- “sasql_query” on page 769
- “sasql_real_query” on page 770
- “sasql_store_result” on page 783
- “sasql_use_result” on page 784

**Result sets**
- “sasql_data_seek” on page 757
- “sasql_fetch_array” on page 760
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_field” on page 761
- “sasql_fetch_object” on page 762
- “sasql_fetch_row” on page 762
- “sasql_field_count” on page 763
- “sasql_free_result” on page 764
- “sasql_num_rows” on page 767
- “sasql_result_all” on page 771

**Transactions**
- “sasql_commit” on page 756
- “sasql_rollback” on page 772
Statements

- “sasql_prepare” on page 768
- “sasql_stmt_affected_rows” on page 773
- “sasql_stmt_bind_param” on page 773
- “sasql_stmt_bind_param_ex” on page 774
- “sasql_stmt_bind_result” on page 775
- “sasql_stmt_close” on page 775
- “sasql_stmt_data_seek” on page 776
- “sasql_stmt_execute” on page 777
- “sasql_stmt_fetch” on page 778
- “sasql_stmt_field_count” on page 778
- “sasql_stmt_free_result” on page 779
- “sasql_stmt_insert_id” on page 779
- “sasql_stmt_next_result” on page 780
- “sasql_stmt_num_rows” on page 780
- “sasql_stmt_param_count” on page 781
- “sasql_stmt_reset” on page 781
- “sasql_stmt_result_metadata” on page 781
- “sasql_stmt_send_long_data” on page 782
- “sasql_stmt_store_result” on page 782

Miscellaneous

- “sasql_escape_string” on page 759
- “sasql_get_client_info” on page 764

**sasql_affected_rows**

Prototype

```c
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.

Related functions

- “sasql_num_rows” on page 767
**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.

Related functions

- “sasql_rollback” on page 772
- “sasql_set_option” on page 772
- “sasql_pconnect” on page 767
- “sasql_disconnect” on page 758

**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 )
```
Establishes a connection to a SQL Anywhere database.

Parameters

$\texttt{con\_str}$  A connection string as recognized by SQL Anywhere.

Returns

A positive SQL Anywhere connection resource on success, or an error and 0 on failure.

Related functions

- “\texttt{sasql\_pconnect}” on page 767
- “\texttt{sasql\_disconnect}” on page 758

See also

- “Connection parameters” [\textit{SQL Anywhere Server - Database Administration}]
- “SQL Anywhere database connections” [\textit{SQL Anywhere Server - Database Administration}]

**sasql\_data\_seek**

Prototype

```php
bool sasql_data_seek( sasql_result $result, int row_num )
```

Description

Positions the cursor on row \textit{row\_num} on the $\texttt{result}$ that was opened using \texttt{sasql\_query}.

Parameters

$\texttt{result}$  The result resource returned by the \texttt{sasql\_query} function.

\texttt{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

\texttt{TRUE} on success or \texttt{FALSE} on error.

Related functions

- “\texttt{sasql\_fetch\_field}” on page 761
- “\texttt{sasql\_fetch\_array}” on page 760
- “\texttt{sasql\_fetch\_assoc}” on page 760
- “\texttt{sasql\_fetch\_row}” on page 762
- “\texttt{sasql\_fetch\_object}” on page 762
- “\texttt{sasql\_query}” on page 769
sasql_disconnect

Prototype

bool sasql_disconnect( sasql_conn $conn )

Description

Closes a connection that has already been opened with sasql_connect.

Parameters

$conn  The connection resource returned by a connect function.

Returns

TRUE on success or FALSE on error.

Related functions

● “sasql_connect” on page 756
● “sasql_pconnect” on page 767

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].

Related functions

● “sasql_errorcode” on page 759
● “sasql_sqlstate” on page 783
● “sasql_set_option” on page 772
● “sasql_stmt_errno” on page 776
● “sasql_stmt_error” on page 777
sasql_errorcode

Prototype

```c
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].

Related functions

- “sasql_connect” on page 756
- “sasql_pconnect” on page 767
- “sasql_error” on page 758
- “sasql_sqlstate” on page 783
- “sasql_set_option” on page 772
- “sasql_stmt_errno” on page 776
- “sasql_stmt_error” on page 777

sasql_escape_string

Prototype

```c
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.

Related functions

- “sasql_real_escape_string” on page 770
- “sasql_connect” on page 756

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.

Related functions

- “sasql_data_seek” on page 757
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_field” on page 761
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762

sasql_fetch_assoc

Prototype

array sasql_fetch_assoc(sasql_result $result)

Related functions

- “sasql_data_seek” on page 757
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_field” on page 761
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762
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.

Related functions
- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_field” on page 761
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762

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 542.
• **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.

**Related functions**

- “sasql_data_seek” on page 757
- “sasql_fetch_array” on page 760
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762

### sasql_fetch_object

**Prototype**

```php
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.

**Related functions**

- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_array” on page 760
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_row” on page 762

### sasql_fetch_row

**Prototype**

```php
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.

Related functions

- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_array” on page 760
- “sasql_fetch_assoc” on page 760
- “sasql_fetch_object” on page 762

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**

```c
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.

**Related functions**

- “sasql_query” on page 769

**sasql_get_client_info**

**Prototype**

```c
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**

```c
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
$sconn 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 console.

Parameters
$conn The connection resource returned by a connect function.
$message A message to be written to the server console.

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.

Related functions
- “sasql_store_result” on page 783
- “sasql_use_result” on page 784
- “sasql_field_count” on page 763

sasql_next_result

Prototype
```c
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.

Related functions
- “sasql_use_result” on page 784
- “sasql_store_result” on page 783

sasql_num_fields

Prototype
```c
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.

Related functions
● “sasql_num_rows” on page 767
● “sasql_query” on page 769

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 772.

Related functions
● “sasql_num_fields” on page 766
● “sasql_query” on page 769

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.

Related functions

- “sasql_connect” on page 756
- “sasql_disconnect” on page 758

See also

- “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.
Related functions

- “sasql_stmt_param_count” on page 781
- “sasql_stmt_bind_param” on page 773
- “sasql_stmt_bind_param_ex” on page 774
- “sasql_prepare” on page 768
- “sasql_stmt_execute” on page 777
- “sasql_connect” on page 756
- “sasql_pconnect” on page 767

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.

Related functions

- “sasql_real_query” on page 770
- “sasql_free_result” on page 764
- “sasql_fetch_array” on page 760
- “sasql_fetch_field” on page 761
- “sasql_fetch_object” on page 762
- “sasql_fetch_row” on page 762
**sasql_real_escape_string**

**Prototype**

```php
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.

**Related functions**

- "sasql_escape_string" on page 759
- "sasql_connect" on page 756

**sasql_real_query**

**Prototype**

```php
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.
Related functions

- “sasql_query” on page 769
- “sasql_store_result” on page 783
- “sasql_use_result” on page 784
- “sasql_field_count” on page 763

sasql_result_all

Prototype

```php
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.

Related functions

- “sasql_query” on page 769
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.

Related functions

● “sasql_commit” on page 756
● “sasql_set_option” on page 772

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_commit</td>
<td>When this option is set to on, the database server commits after executing each statement.</td>
<td>on</td>
</tr>
<tr>
<td>row_counts</td>
<td>When this option is set to FALSE, the sasql_num_rows function returns an estimate of the number of rows affected. If you want to obtain an exact count, set this option to TRUE.</td>
<td>FALSE</td>
</tr>
<tr>
<td>verbose_errors</td>
<td>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.</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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.

Related functions
- “sasql_commit” on page 756
- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_num_rows” on page 767
- “sasql_rollback” on page 772

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.

Related functions
- “sasql_stmt_execute” on page 777

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.

Related functions

- “sasql_prepare” on page 768
- “sasql_stmt_param_count” on page 781
- “sasql_stmt_bind_param_ex” on page 774
- “sasql_stmt_execute” on page 777

sasql_stmt_bind_param_ex

Prototype

```php
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 1 and sasql_stmt_param_count.
- **$var** A PHP variable. Only references to PHP variables are allowed.
- **$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.
Related functions

- “sasql_prepare” on page 768
- “sasql_stmt_param_count” on page 781
- “sasql_stmt_bind_param” on page 773
- “sasql_stmt_execute” on page 777

sasql_stmt_bind_result

Prototype

```php
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.

Related functions

- “sasql_stmt_execute” on page 777
- “sasql_stmt_fetch” on page 778

sasql_stmt_close

Prototype

```php
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.
Related functions

- “sasql_stmt_result_metadata” on page 781
- “sasql_prepare” on page 768

`sasql_stmt_data_seek`

Prototype

```php
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 minus 1.

Returns

TRUE on success or FALSE failure.

Related functions

- “sasql_stmt_num_rows” on page 780

`sasql_stmt_errno`

Prototype

```php
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].
Related functions

- “sasql_stmt_error” on page 777
- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_prepare” on page 768
- “sasql_stmt_result_metadata” on page 781

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].

Related functions

- “sasql_stmt_errno” on page 776
- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_prepare” on page 768
- “sasql_stmt_result_metadata” on page 781

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.

Related functions

- “sasql_prepare” on page 768
- “sasql_stmt_param_count” on page 781
- “sasql_stmt_bind_param” on page 773
- “sasql_stmt_bind_param_ex” on page 774
- “sasql_stmt_result_metadata” on page 781
- “sasql_stmt_bind_result” on page 775

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.

Related functions

- “sasql_stmt_bind_result” on page 775

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.

Related functions
● “sasql_stmt_result_metadata” on page 781

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.

Related functions
● “sasql_stmt_execute” on page 777
  ● “sasql_stmt_store_result” on page 782

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.
Related functions

- “sasql_stmt_execute” on page 777

`sasql_stmt_next_result`

Prototype

```php
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.

Related functions

- “sasql_stmt_result_metadata” on page 781

`sasql_stmt_num_rows`

Prototype

```php
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 `sasql_stmt_store_result` was called on.

Returns

- The number of rows available in the result or 0 on failure.

Related functions

- “sasql_stmt_execute” on page 777
- “sasql_stmt_store_result” on page 782
**sasql_stmt_param_count**

**Prototype**

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

**Description**

Returns the number of parameters in the supplied prepared statement handle.

**Parameters**

- `$stmt` A statement resource returns by the `sasql_prepare` function.

**Returns**

The number of parameters or FALSE on error.

**Related functions**

- “sasql_prepare” on page 768
- “sasql_stmt_bind_param” on page 773
- “sasql_stmt_bind_param_ex” on page 774

**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.

**Related functions**

- “sasql_stmt_send_long_data” on page 782

**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() - 1).
$ data     The data to be sent.

Returns
TRUE on success or FALSE on failure.

Related functions
● “sasql_stmt_bind_param” on page 773
● “sasql_stmt_bind_param_ex” on page 774
● “sasql_prepare” on page 768
● “sasql_stmt_param_count” on page 781

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.

**Related functions**

- “sasql_stmt_free_result” on page 779
- “sasql_stmt_execute” on page 777

---

**sasql_store_result**

**Prototype**

```php
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.

**Related functions**

- “sasql_data_seek” on page 757
- “sasql_stmt_execute” on page 777

---

**sasql_sqlstate**

**Prototype**

```php
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].

Related functions

- “sasql_error” on page 758
- “sasql_errorcode” on page 759

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.

Related functions

- “sasql_data_seek” on page 757
- “sasql_stmt_execute” on page 777

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_.

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
sqlanywhere_commit (deprecated)

Prototype

```php
bool sqlanywhere_commit( resource link_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead: “sasql_commit” on page 756.

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.

```php
$result = sqlanywhere_commit( $conn );
```

Related functions

- “sasql_commit” on page 756
- “sasql_pconnect” on page 767
- “sasql_disconnect” on page 758
- “sqlanywhere_pconnect (deprecated)” on page 797
- “sqlanywhere_disconnect (deprecated)” on page 787

sqlanywhere_connect (deprecated)

Prototype

```php
resource sqlanywhere_connect( string con_str )
```

Description

This function is deprecated. You should use the following PHP function instead: “sasql_connect” on page 756.

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.

```php
$conn = sqlanywhere_connect( "UID=DBA;PWD=sql" );
```

Related functions

- “sasql_connect” on page 756
- “sasql_pconnect” on page 767
- “sasql_disconnect” on page 758
- “sqlanywhere_pconnect (deprecated)” on page 797
- “sqlanywhere_disconnect (deprecated)” on page 787

sqlanywhere_data_seek (deprecated)

Prototype

```php
bool sqlanywhere_data_seek( resource result_identifier, int row_num )
```

Description

This function is deprecated. You should use the following PHP function instead:

“sasql_data_seek” on page 757.

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.

```php
sqlanywhere_data_seek( $result, 5 );
```
Related functions

- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_array” on page 760
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762
- “sasql_query” on page 769
- “sqlanywhere_fetch_field (deprecated)” on page 791
- “sqlanywhere_fetch_array (deprecated)” on page 790
- “sqlanywhere_fetch_row (deprecated)” on page 793
- “sqlanywhere_fetch_object (deprecated)” on page 792
- “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_disconnect (deprecated)

Prototype

```php
bool sqlanywhere_disconnect( resource link_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_disconnect” on page 758.

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.

```php
sqlanywhere_disconnect( $conn );
```

Related functions

- “sasql_disconnect” on page 758
- “sasql_connect” on page 756
- “sasql_pconnect” on page 767
- “sqlanywhere_connect (deprecated)” on page 785
- “sqlanywhere_pconnect (deprecated)” on page 797
sqlanywhere_error (deprecated)

Prototype

string sqlanywhere_error( [ resource link_identifier ] )

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_error” on page 758.

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.

```php
$result = sqlanywhere_query( $conn, "SELECT * FROM table_that_does_not_exist" );
if( ! $result ) {
    $error_msg = sqlanywhere_error( $conn );
    echo "Query failed. Reason: $error_msg";
}
```

Related functions

- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_set_option” on page 772
- “sqlanywhere_errorcode (deprecated)” on page 788
- “sqlanywhere_set_option (deprecated)” on page 800

sqlanywhere_errorcode (deprecated)

Prototype

bool sqlanywhere_errorcode( [ resource link_identifier ] )
Description

This function is deprecated. You should use the following PHP function instead:
“sasql_errorcode” on page 759.

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.

```php
$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";
}
```

Related functions

● “sasql_error” on page 758
● “sasql_set_option” on page 772
● “sqlanywhere_error (deprecated)” on page 788
● “sqlanywhere_set_option (deprecated)” on page 800

sqlanywhere_execute (deprecated)

Prototype

```php
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.

```php
if( sqlanywhere_execute( $conn, "CREATE TABLE my_test_table( INT id )" ) ) {
    // handle success
} else {
    // handle failure
}
```

Related functions

- “sasql_query” on page 769
- “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_fetch_array (deprecated)

Prototype

```php
array sqlanywhere_fetch_array( resource result_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead:

“sasql_fetch_array” on page 760.

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.

```php
$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";
}
```
Related functions

- “sasql_fetch_array” on page 760
- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762
- “sasql_query” on page 769
- “sqlanywhere_data_seek (deprecated)” on page 786
- “sqlanywhere_fetch_field (deprecated)” on page 791
- “sqlanywhere_fetch_row (deprecated)” on page 793
- “sqlanywhere_fetch_object (deprecated)” on page 792
- “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_fetch_field (deprecated)

Prototype

object sqlanywhere_fetch_field(  resource result_identifier [, field_offset ] )

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_fetch_field” on page 761.

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 
";
    echo " Field name = $field->name 
";
}

Related functions
● “sasql_data_seek” on page 757
● “sasql_fetch_field” on page 761
● “sasql_fetch_array” on page 760
● “sasql_fetch_row” on page 762
● “sasql_fetch_object” on page 762
● “sasql_query” on page 769
● “sqlanywhere_data_seek (deprecated)” on page 786
● “sqlanywhere_fetch_array (deprecated)” on page 790
● “sqlanywhere_fetch_row (deprecated)” on page 793
● “sqlanywhere_fetch_object (deprecated)” on page 792
● “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_fetch_object (deprecated)

Prototype
object sqlanywhere_fetch_object( resource result_identifier )

Description
This function is deprecated. You should use the following PHP function instead:
“sasql_fetch_object” on page 762.

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 
";  # output the data in the first column only.
    }
Related functions

- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_array” on page 760
- “sasql_fetch_row” on page 762
- “sasql_fetch_object” on page 762
- “sasql_query” on page 769
- “sqlanywhere_query (deprecated)” on page 798
- “sqlanywhere_data_seek (deprecated)” on page 786
- “sqlanywhere_fetch_field (deprecated)” on page 791
- “sqlanywhere_fetch_array (deprecated)” on page 790
- “sqlanywhere_fetch_row (deprecated)” on page 793

sqlanywhere_fetch_row (deprecated)

Prototype

```php
array sqlanywhere_fetch_row( resource result_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_fetch_row” on page 762.

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.

```php
while( ($row = sqlanywhere_fetch_row( $result ) ) ) {
    echo "$row[0] \n"; # output the data in the first column only.
}
```
Related functions

- “sasql_fetch_row” on page 762
- “sasql_data_seek” on page 757
- “sasql_fetch_field” on page 761
- “sasql_fetch_array” on page 760
- “sasql_fetch_object” on page 762
- “sasql_query” on page 769
- “sqlanywhere_data_seek (deprecated)” on page 786
- “sqlanywhere_fetch_field (deprecated)” on page 791
- “sqlanywhere_fetch_array (deprecated)” on page 790
- “sqlanywhere_fetch_object (deprecated)” on page 792
- “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_free_result (deprecated)

Prototype

```php
bool sqlanywhere_free_result( resource result_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_free_result” on page 764.

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.

```php
sqlanywhere_free_result( $result );
```

Related functions

- “sasql_query” on page 769
- “sasql_free_result” on page 764
- “sqlanywhere_query (deprecated)” on page 798
sqlanywhere_identity (deprecated)

Prototype

```c
int sqlanywhere_identity( resource link_identifier )
```

```c
int sqlanywhere_insert_id( resource link_identifier )
```

Description

This function is deprecated. You should use the following PHP function instead:

"sasql_insert_id" on page 764.

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.

```php
if( sqlanywhere_execute( $conn, "INSERT INTO my_auto_increment_table VALUES ( 1 )" ) ) {
    $insert_id = sqlanywhere_insert_id( $conn );
    echo "Last insert id = $insert_id";
}
```

Related functions

- "sasql_insert_id” on page 764
- “sqlanywhere_execute (deprecated)” on page 789

sqlanywhere_num_fields (deprecated)

Prototype

```c
int sqlanywhere_num_fields( resource result_identifier )
```
Description

This function is deprecated. You should use the following PHP function instead:
“sasql_field_count” on page 763.

Returns the number of columns (fields) the \texttt{result_identifier} contains.

Parameters

\texttt{result_identifier} \quad The result resource returned by the sqlanywhere_query function.

Returns

A positive number of columns, or an error if \texttt{result_identifier} is not valid.

Example

This example returns a value indicating how many columns are in the result set.

\begin{verbatim}
$num_columns = sqlanywhere_num_fields( $result );
\end{verbatim}

Related functions

- “sasql_field_count” on page 763
- “sasql_query” on page 769
- “sqlanywhere_query (deprecated)” on page 798

\textbf{sqlanywhere_num_rows (deprecated)}

Prototype

\begin{verbatim}
int sqlanywhere_num_rows( resource \texttt{result_identifier} )
\end{verbatim}

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_num_rows” on page 767.

Returns the number of rows that the \texttt{result_identifier} contains.

Parameters

\texttt{result_identifier} \quad 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 772.

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
}

Related functions

- “sasql_num_rows” on page 767
- “sasql_query” on page 769
- “sqlanywhere_query (deprecated)” on page 798

sqlanywhere_pconnect (deprecated)

Prototype

resource sqlanywhere_pconnect( string con_str )

Description

This function is deprecated. You should use the following PHP function instead: “sasql_pconnect” on page 767.

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" );

Related functions

- “sasql_pconnect” on page 767
- “sasql_connect” on page 756
- “sasql_disconnect” on page 758
- “sqlanywhere_connect (deprecated)” on page 785
- “sqlanywhere_disconnect (deprecated)” on page 787
sqlanywhere_query (deprecated)

Prototype

    resource sqlanywhere_query( resource link_identifier, string sql_str )

Description

This function is deprecated. You should use the following PHP function instead:
“sasql_query” on page 769.

Prepares and executes the SQL query \texttt{sql\_str} on the connection identified by \texttt{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

- \texttt{link\_identifier}  The link identifier returned by the sqlanywhere\_connect function.
- \texttt{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" );

Related functions

- “sasql_query” on page 769
- “sasql\_free\_result” on page 764
- “sasql\_fetch\_array” on page 760
- “sasql\_fetch\_field” on page 761
- “sasql\_fetch\_object” on page 762
- “sasql\_fetch\_row” on page 762
- “sqlanywhere\_execute (deprecated)” on page 789
- “sqlanywhere\_free\_result (deprecated)” on page 794
- “sqlanywhere\_fetch\_array (deprecated)” on page 790
- “sqlanywhere\_fetch\_field (deprecated)” on page 791
- “sqlanywhere\_fetch\_object (deprecated)” on page 792
- “sqlanywhere\_fetch\_row (deprecated)” on page 793
sqlanywhere_result_all (deprecated)

Prototype

```php
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.

```php
$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.

```php
$result = sqlanywhere_query( $conn, "SELECT GivenName, Surname FROM Employees" );
```
sqlanywhere_rollback (deprecated)

**Prototype**

```php
bool sqlanywhere_rollback( resource link_identifier )
```

**Description**

This function is deprecated. You should use the following PHP function instead:
“sasql_rollback” on page 772.

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.

```php
$result = sqlanywhere_rollback( $conn );
```

**Related functions**

- “sasql_rollback” on page 772
- “sasql_commit” on page 756
- “sasql_set_option” on page 772
- “sqlanywhere_commit (deprecated)” on page 785
- “sqlanywhere_set_option (deprecated)” on page 800

sqlanywhere_set_option (deprecated)

**Prototype**

```php
bool sqlanywhere_set_option( resource link_identifier, string option, mixed value )
```
Description

This function is deprecated. You should use the following PHP function instead:
“sasql_set_option” on page 772.

Sets the value of the specified option on the specified connection. You can set the value for the following options:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_commit</td>
<td>When this option is set to on, the database server commits after executing each statement.</td>
<td>on</td>
</tr>
<tr>
<td>row_counts</td>
<td>When this option is set to FALSE, the sqlanywhere_num_rows function returns an estimate of the number of rows affected. If you want to obtain an exact count, set this option to TRUE.</td>
<td>FALSE</td>
</tr>
<tr>
<td>verbose_errors</td>
<td>When this option is set to TRUE, the PHP driver returns verbose errors. When this option is set to FALSE, you must call the sqlanywhere_error or sqlanywhere_errorcode functions to get further error information.</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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.

```php
$result = sqlanywhere_set_option( $conn, "auto_commit", "Off" );
$result = sqlanywhere_set_option( $conn, "auto_commit", 0 );
$result = sqlanywhere_set_option( $conn, "auto_commit", False );
```
Related functions

- “sasql_set_option” on page 772
- “sasql_commit” on page 756
- “sasql_error” on page 758
- “sasql_errorcode” on page 759
- “sasql_num_rows” on page 767
- “sasql_rollback” on page 772
- “sqlanywhere_commit (deprecated)” on page 785
- “sqlanywhere_error (deprecated)” on page 788
- “sqlanywhere_errorcode (deprecated)” on page 788
- “sqlanywhere_num_rows (deprecated)” on page 796
- “sqlanywhere_rollback (deprecated)” on page 800
Building the SQL Anywhere PHP module on Unix and Mac OS X

To connect PHP to SQL Anywhere using the SQL Anywhere PHP module on Unix and Mac OS X, you must add the SQL Anywhere PHP module'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 the steps detailed in this document:

- You must have a SQL Anywhere installation which may run on the same computer as the Apache web server, or on a different computer.
- The source code for the SQL Anywhere PHP module is available for download at http://download.sybase.com/ianywhere/php/2.0.3/src/sasql_php.zip. You also need sqlpp and libdblib11.so (Unix) or libdblib11.dylib (Mac OS X) installed (check your SQL Anywhere lib32 directory).
- You will require the PHP source code, which can be downloaded from http://www.php.net. Version 5.2.6 of PHP is a recent stable release.
- You will require 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 module, you need to have libdbodbc11.so (Unix) or libdbodbc11.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 module files to the PHP source tree**


2. From the directory where you saved the SQL Anywhere PHP module, extract the files to the `ext` subdirectory of the PHP source tree (Mac OS X users should replace `tar` with `gnutar`):

   ```bash
   $ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
   
   For example:
   
   $ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.6/ext
   
   3. Make PHP aware of the module:

   ```bash
   $ cd PHP-source-directory/ext/sqlanywhere
   $ touch *
   $ cd ~/PHP-source-directory
   $ ./buildconf
   
   The following example is for PHP version 5.2.6. You must change `php-5.2.6` to the version of PHP you are using.

   ```bash
   $ cd ~/php-5.2.6/ext/sqlanywhere
   $ touch *
   $ cd ~/php-5.2.6
   $ ./buildconf
   
   4. Verify that PHP is aware of the module:

   ```bash
   $ ./configure --help | egrep sqlanywhere
   
   If you were successful in making PHP aware of the SQL Anywhere module, you should see the following text:

   ```bash
   --with-sqlanywhere=[DIR]
   
   If you are unsuccessful, keep track of the output of this command and post it to the sybase.public.sqlanywhere.linux newsgroup 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.
For information about compiling PHP as an Apache module, see “Compiling PHP as an Apache module” on page 805.

For information about compiling PHP as a CGI executable, see “Compiling PHP as a CGI executable” on page 807.

Compiling PHP as an Apache module

The first two steps in the following instructions 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 preinstalled Apache web server.

To compile PHP as an Apache module

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/sqlanywhere11`). On Mac OS X, the default directory is `/Applications/SQLAnywhere11/System`.

<table>
<thead>
<tr>
<th>If you are using this shell...</th>
<th>...use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh, ksh, bash</td>
<td>./bin32/sa_config.sh</td>
</tr>
<tr>
<td>csh, tcsh</td>
<td>source ./bin32/sa_config.csh</td>
</tr>
</tbody>
</table>

4. Configure PHP as an Apache module to include the SQL Anywhere PHP module.

   Execute the following commands:
The following example is for PHP version 5.2.6. You must change `php-5.2.6` to the version of PHP you are using.

```
$ cd ~/php-5.2.6
$ ./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/sqlanywhere11
checking SQL Anywhere version... 11
```

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 `libdblib11.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.

For information about testing and using your setup, see “Running PHP test scripts in your web pages” on page 746.

## Compiling PHP as a CGI executable

**To 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 805.

2. Configure PHP as a CGI executable and with the SQL Anywhere PHP module.
   - Execute the following command from the directory where your PHP files were extracted:
     ```
     $ cd PHP-source-directory
     $ ./configure --with-sqlanywhere
     ```
   - For example:
     ```
     $ cd ~/php-5.2.6/
     $ ./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/sqlanywhere10
     checking SQL Anywhere version... 9
     ```

3. Compile the executable:
   ```
   $ make
   ```

4. Install the components.
   ```
   $ make install
   ```
For information about testing and using PHP, see “Running PHP test scripts in your web pages” on page 746.
SQL Anywhere for Ruby

Contents

Ruby support in SQL Anywhere ................................................................................ 810
Rails support in SQL Anywhere ................................................................................. 812
Ruby-DBI Driver for SQL Anywhere .......................................................................... 814
SQL Anywhere Ruby API .......................................................................................... 818
Ruby support in SQL Anywhere

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, please visit the following site: http://rubyforge.org/projects/rubygems/. We recommend that you install version 1.3.1 or later.

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

Note that this package is a prerequisite for any of the other SQL Anywhere Ruby packages. For more information, see:

- “SQL Anywhere Ruby API” on page 818
- 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

For more information, see:

- “Rails support in SQL Anywhere” on page 812
- 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 more information, see:

- “Ruby-DBI Driver for SQL Anywhere” on page 814
- Download source (http://rubyforge.org/projects/sqlanywhere)
- RDocs (http://sqlanywhere.rubyforge.org/dbd-sqlanywhere)
- Ruby/DBI - Direct database access layer for Ruby (http://ruby-db.rubyforge.org/)

For feedback on any of these packages, please use the sqlanywhere-users@rubyforge.com mailing list. For general questions about using SQL Anywhere in a web environment, please use the SQL Anywhere Web Development forum. For general questions on SQL Anywhere and its usage, please use the sybase.public.sqlanywhere.general newsgroup.
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. You might consider including the “SQL Anywhere Ruby API” on page 818 as part of your familiarization with Ruby.

If you are ready to jump into Rails development, there are a few things you need to do.

Prerequisites

- **RubyGems** You should install RubyGems. It makes installation of Ruby packages so much easier. At the time of writing, version 1.3.1 was required for Rails development. The Ruby on Rails (http://www.rubyonrails.org/) web site will direct you to the correct version to install.

- **Ruby** You will need to install the Ruby interpreter on your system. The Ruby on Rails (http://www.rubyonrails.org/) web site recommends which version to install.

- **Rails** With RubyGems, you can install all of Rails and its dependencies with a single command line:

  ```
gem install rails
  ```

- **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.

1. 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 2.2.2 of Rails, then the path to this file would be `\Ruby\lib\ruby\gems\1.8\gems\rails-2.2.2\configs\databases`. The contents of this file should be:

   ```
   # SQL Anywhere database configuration
   #
   # This configuration file defines the pattern 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
     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
  database: <%= app_name %>_test
  username: DBA
  password: sql

production:
  adapter: sqlanywhere
  database: <%= app_name %>_production
  username: DBA
  password: sql

2. You must update the Rails app_generator.rb file. Using the same assumptions in step 1 above, this file is located in the path \Ruby\lib\ruby\gems\1.8\gems\rails-2.2.2\lib\rails_generator\generators\applications\app. Edit the app_generator.rb file and locate the following line:

   DATABASES = %w(mysql oracle postgresql sqlite2 sqlite3 frontbase ibm_db)

Add sqlanywhere to the list as follows.

   DATABASES = %w(sqlanywhere mysql oracle postgresql sqlite2 sqlite3 frontbase ibm_db)

If you want, you can also change the DEFAULT_DATABASE setting (on the next line) to read as follows:

   DEFAULT_DATABASE = 'sqlanywhere'

Now save the file and exit.

Learning Rails

We recommend that you start with the excellent Getting Started With Rails tutorial on the Ruby on Rails website. 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 blog -d sqlanywhere

If you changed the DEFAULT_DATABASE setting, then the -d sqlanywhere option is not required.

Also, note that the blog tutorial requires that you set up three databases. After you have initialized the project, you can change to the root directory of the project and create three databases as follows.

   dbinit blog_development
dbinit blog_test
dbinit blog_production

Before you continue, you must start the database server and the three databases as follows.

   dbsrv11 blog_development.db blog_production.db blog_test.db

You are now ready to explore Ruby on Rails web development using the tutorial.

For more information about the Ruby on Rails web development framework, visit the Ruby on Rails (http://www.rubyonrails.org/) web site.
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.

```ruby
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:

```ruby
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password, options)
```

- **server-name** is the name of the database server that you wish 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.

```ruby
require 'dbi'
DBI.connect('DBI:SQLAnywhere:demo', 'DBA', 'sql') do |dbh|
  if dbh.ping
    print "Successfully Connected\n"
  end
  dbh.disconnect()
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:
Because the user ID and password are specified in the connection string, it is not necessary to specify parameters 2 and 3 to the connect function. However, if you pass a hash of additional connection parameters, then specify an empty string (' ') for the user ID and password parameters.

The following example shows how additional connection parameters can be passed to the connect function as a hash of key/value pairs.

```ruby
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"
  end
  dbh.disconnect()
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.

```ruby
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
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
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
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. See “max_cursor_count option [database]” [SQL Anywhere Server - Database Administration], and “max_statement_count option [database]” [SQL Anywhere Server - Database Administration].

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 place holders 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)
    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
SQL Anywhere Ruby API

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 demo 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 11 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” on page 823

Example

    affected = api.sqlany_affected( stmt )

sqlany_bind_param

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” on page 822

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

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” on page 832

Example

api.sqlany_clear_error( conn )

sqlany_client_version

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

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” on page 835

Example
```ruby
rc = api.sqlany_commit( conn )
```

**sqlany_connect**

Creates a connection to a SQL Anywhere database server using the specified connection object and
connection string.

Syntax
```ruby
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” on page 832
- “sqlany_error” on page 823
- “Connection parameters” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere database connections” [SQL Anywhere Server - Database Administration]

Example
```ruby
# 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

Describes the bind parameters of a prepared statement.

Syntax

\texttt{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” on page 819
- “sqlany\_prepare” on page 834

Example

\begin{verbatim}
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(), \
print "Param dir = ", param.get\_direction(), \
param.set\_value(50)
rc = api.sqlany\_bind\_param( stmt, 0, param )
\end{verbatim}

sqlany_disconnect

Disconnects a SQL Anywhere connection. All uncommitted transactions are rolled back.

Syntax

\texttt{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” on page 821
- “sqlany_new_connection” on page 832

Example

```ruby
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

sqlany_error

Returns the last error code and message stored in the connection object.

Syntax

```ruby
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” on page 821
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

Example

```ruby
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

sqlany_execute

Executes a prepared statement.

Syntax

```ruby
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” on page 834

Example

```ruby
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

Executes the SQL statement specified by the string argument.

Syntax

```ruby
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” on page 825
- “sqlany_fetch_next” on page 826
- “sqlany_num_cols” on page 832
- “sqlany_get_column” on page 829
Example

```ruby
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**

Executes the specified SQL statement immediately without returning a result set. It is useful for statements that do not return result sets.

**Syntax**

```ruby
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” on page 823

**Example**

```ruby
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
SET Contacts.ID = Contacts.ID + 1000
WHERE Contacts.ID >= 50" )
```

**sqlany_fetch_absolute**

Moves the current row in the result set to the row number specified and then fetches the data at that row.

**Syntax**

```ruby
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” on page 823
- “sqlany_execute” on page 823
- “sqlany_execute_direct” on page 824
- “sqlany_fetch_next” on page 826

Example

```ruby
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

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

```
sqalny_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” on page 823
- “sqlany_execute” on page 823
- “sqlany_execute_direct” on page 824
- “sqlany_fetch_absolute” on page 825

Example

```ruby
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, 

sqlany_fini

Frees resources allocated by the API.

Syntax
   sqlany_fini()

Returns
   Returns nil.

See also
   ● “sqlany_init” on page 831

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

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” on page 832
Example

```ruby
# 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**

Frees resources associated with a statement object.

**Syntax**

```ruby
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" on page 834
- "sqlany_execute_direct" on page 824

**Example**

```ruby
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**

Retrieves information about the parameters that were bound using sqlany_bind_param.

**Syntax**

```ruby
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” on page 819
- “sqlany_describe_bind_param” on page 822
- “sqlany_prepare” on page 834

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

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” on page 823
- “sqlany_execute_direct” on page 824
- “sqlany_fetch_absolute” on page 825
- “sqlany_fetch_next” on page 826

Example

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
```
SQL Anywhere for Ruby

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
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 sqlany_num_cols() - 1).

2

String

Column name.

3

Integer

Column type. See “Column types” on page 836.

4

Integer

Column native type. See “Native column types” on page 836.

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” on page 823
● “sqlany_execute_direct” on page 824
● “sqlany_prepare” on page 834

830

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1


**Example**

```ruby
# 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**

Advances to the next result set in a multiple result set query.

**Syntax**

```ruby
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” on page 823
- “sqlany_execute_direct” on page 824

**Example**

```ruby
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**

Initializes the interface.

**Syntax**

```ruby
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” on page 827
Example

```ruby
# 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`

Creates a connection object.

Syntax

```ruby
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” on page 821
- “sqlany_disconnect” on page 822

Example

```ruby
# 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`

Returns number of columns in the result set.

Syntax

```ruby
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” on page 823
- “sqlany_execute_direct” on page 824
- “sqlany_prepare” on page 834

Example

```ruby
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

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” on page 834

Example

```ruby
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

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 [database]” [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” on page 823
- “sqlany_execute_direct” on page 824

Example

```ruby
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

Prepares the supplied SQL string

Syntax

```ruby
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” on page 823
- “sqlany_free_stmt” on page 828
- “sqlany_num_params” on page 833
- “sqlany_describe_bind_param” on page 822
- “sqlany_bind_param” on page 819

Example

```ruby
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

Rolls back the current transaction.

Syntax

```ruby
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” on page 820

Example

```ruby
rc = api.sqlany_rollback(conn)
```

sqlany_sqlstate

Retrieves the current SQL state.

Syntax

```ruby
sqlany_sqlstate($conn)
```

Parameters

- `$conn` A connection object returned from sqlany_new_connection.
Returns

Returns a scalar value that is the current 5-character SQL state.

See also

- “sqlany_error” on page 823
- “SQL Anywhere error messages sorted by SQLSTATE”

Example

```ruby
sql_state = api.sqlany_sqlstate( conn )
```

## Column types

The following Ruby class defines the column types returned by some SQL Anywhere Ruby functions.

```ruby
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
  A_VAL8         = 10
  A_UVAL8        = 11
end
```

## Native column types

The following table defines the native column types returned by some SQL Anywhere functions.

<table>
<thead>
<tr>
<th>Native Type Value</th>
<th>Native Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>384</td>
<td>DT_DATE</td>
</tr>
<tr>
<td>388</td>
<td>DT_TIME</td>
</tr>
<tr>
<td>390</td>
<td>DT_TIMESTAMP_STRUCT</td>
</tr>
<tr>
<td>392</td>
<td>DT_TIMESTAMP</td>
</tr>
<tr>
<td>448</td>
<td>DT_VARCHAR</td>
</tr>
<tr>
<td>452</td>
<td>DT_FIXCHAR</td>
</tr>
<tr>
<td>456</td>
<td>DT_LONGVARCHAR</td>
</tr>
<tr>
<td>460</td>
<td>DT_STRING</td>
</tr>
<tr>
<td>Native Type Value</td>
<td>Native Type</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>480</td>
<td>DT_DOUBLE</td>
</tr>
<tr>
<td>482</td>
<td>DT_FLOAT</td>
</tr>
<tr>
<td>484</td>
<td>DT_DECIMAL</td>
</tr>
<tr>
<td>496</td>
<td>DT_INT</td>
</tr>
<tr>
<td>500</td>
<td>DT_SMALLINT</td>
</tr>
<tr>
<td>524</td>
<td>DT_BINARY</td>
</tr>
<tr>
<td>528</td>
<td>DT_LONGBINARY</td>
</tr>
<tr>
<td>600</td>
<td>DT_VARIABLE</td>
</tr>
<tr>
<td>604</td>
<td>DT_TINYINT</td>
</tr>
<tr>
<td>608</td>
<td>DT_BIGINT,</td>
</tr>
<tr>
<td>612</td>
<td>DT_UNSINT</td>
</tr>
<tr>
<td>616</td>
<td>DT_UNSSMALLINT</td>
</tr>
<tr>
<td>620</td>
<td>DT_UNSBIGINT</td>
</tr>
<tr>
<td>624</td>
<td>DT_BIT</td>
</tr>
<tr>
<td>628</td>
<td>DT_NSTRING</td>
</tr>
<tr>
<td>632</td>
<td>DT_NFIXCHAR</td>
</tr>
<tr>
<td>636</td>
<td>DT_NVARCHAR</td>
</tr>
<tr>
<td>640</td>
<td>DT_LONGNVARCHAR</td>
</tr>
</tbody>
</table>
Sybase Open Client API

Contents

Open Client architecture ................................................................. 840
What you need to build Open Client applications ......................... 841
Data type mappings ......................................................................... 842
Using SQL in Open Client applications ........................................... 844
Known Open Client limitations of SQL Anywhere ......................... 847
Open Client architecture

**Note**
This chapter 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 chapter 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 programmers. 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.

For more information about running Open Client applications with SQL Anywhere, see “Using 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 runtimes 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.

<table>
<thead>
<tr>
<th>SQL Anywhere data type</th>
<th>Open Client data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>int</td>
</tr>
<tr>
<td>unsigned int</td>
<td>bigint</td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>numeric(20,0)</td>
</tr>
<tr>
<td>date</td>
<td>smalldatetime</td>
</tr>
<tr>
<td>time</td>
<td>smalldatetime</td>
</tr>
<tr>
<td>string</td>
<td>varchar</td>
</tr>
<tr>
<td>timestamp</td>
<td>datetime</td>
</tr>
</tbody>
</table>

**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.

In most cases, the Open Client data type is mapped to a SQL Anywhere data type that has 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 mappings

<table>
<thead>
<tr>
<th>Data type</th>
<th>Open Client lower range</th>
<th>Open Client upper range</th>
<th>SQL Anywhere lower range</th>
<th>SQL Anywhere upper range</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONEY</td>
<td>-922 377 203 685 477.5808</td>
<td>922 377 203 685 477.5807</td>
<td>-1e15 + 0.0001</td>
<td>1e15 - 0.0001</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>-214 748.3648</td>
<td>214 748.3647</td>
<td>-214 748.3648</td>
<td>214 748.3647</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Jan 1, 1753</td>
<td>Dec 31, 9999</td>
<td>Jan 1, 0001</td>
<td>Dec 31, 9999</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>Jan 1, 1900</td>
<td>June 6, 2079</td>
<td>March 1, 1600</td>
<td>Dec 31, 7910</td>
</tr>
</tbody>
</table>

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

The SQL Anywhere implementation of the Open Client TIMESTAMP data type, when such a value is passed in SQL Anywhere, is different from that of Adaptive Server Enterprise. In SQL Anywhere, the value is mapped to the SQL Anywhere DATETIME data type. The default value is NULL in SQL Anywhere and no guarantee is made of its uniqueness. By contrast, Adaptive Server Enterprise ensures that the value is monotonically increasing in value, and so, is unique.

By contrast, the SQL Anywhere TIMESTAMP data type contains year, month, day, hour, minute, second, and fraction of second information. In addition, the DATETIME data type has a greater range of possible values than the Open Client data types that are mapped to it by SQL Anywhere.
Using 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 an introduction to the concepts, see “Using SQL in applications” on page 23. For a complete description, see your Open Client documentation.

Executing SQL statements

You send SQL statements to a database by including them in Client Library function calls. For example, the following pair of calls executes a DELETE statement:

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

For more information on Open Client functions, see Open Client 15.0 Client-Library/C Reference Manual.

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.

To 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.

Using cursors

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.

To 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.

To 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.
Describing query results 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.
SQL Anywhere web services

Contents

Introduction to web services ................................................................. 850
Quick start to web services ................................................................. 851
Creating web services ......................................................................... 856
Starting a database server that listens for web requests ......................... 859
Understanding how URLs are interpreted ............................................. 861
Creating SOAP and DISH web services .............................................. 865
Tutorial: Accessing web services from Microsoft .NET ......................... 868
Tutorial: Accessing web services from JAX-WS ................................ 871
Using procedures that provide HTML documents ................................. 877
Working with data types ...................................................................... 880
Tutorial: Using data types with Microsoft .NET .................................. 887
Tutorial: Using data types with JAX-WS ............................................ 892
Using the iAnywhere WSDL compiler ............................................... 898
Creating web service client functions and procedures ......................... 900
Working with return values and result sets ......................................... 905
Selecting from result sets ................................................................... 907
Using parameters ............................................................................... 908
Working with structured data types .................................................... 911
Working with variables ...................................................................... 917
Working with HTTP headers .............................................................. 919
Using SOAP services ......................................................................... 922
Working with SOAP headers .............................................................. 925
Working with MIME types .................................................................. 932
Using HTTP sessions .......................................................................... 935
Using automatic character set conversion .......................................... 941
Handling errors .................................................................................. 942
Introduction to web services

SQL Anywhere contains a built-in HTTP server that allows you to provide web services, and access web services in other SQL Anywhere databases and standard web services available over the Internet. SOAP is the standard used for this purpose, but the built-in HTTP server in SQL Anywhere also lets you handle standard HTTP and HTTPS requests from client applications.

The term web service has been used to mean a variety of things. Commonly, it refers to software that assists inter-computer data transfer and interoperability. Essentially, web services make segments of business logic available over the Internet. Simple Object Access Protocol (SOAP) is a simple XML-based protocol to let applications exchange information over HTTP.

SOAP provides a method for communication, using the Internet, between applications such as those written in Java or a Microsoft .NET language like Visual C#. SOAP messages define the services that a server provides. Actual data transfer generally takes place using HTTP to exchange XML documents structured in a manner that efficiently encodes the relevant information. Any application, such as a client or server, that participates in SOAP communication is called a SOAP node or SOAP endpoint. Such applications can transmit, receive, or process SOAP messages. You can create SOAP nodes with SQL Anywhere.

For more information about the SOAP standards, see http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

Web services and SQL Anywhere

In the context of SQL Anywhere, the term web services means that SQL Anywhere has the ability to listen for and handle standard SOAP requests. Web services in SQL Anywhere provide client applications an alternative to such traditional interfaces as JDBC and ODBC. Web services can be accessed from client applications written in a variety of languages and running on a variety of platforms. Even common scripting languages such as Perl and Python provide access to web services. You create web services in a database using the CREATE SERVICE statement.

SQL Anywhere can also function as a SOAP or HTTP client, permitting applications running within the database to access standard web services available over the Internet, or provided by other SQL Anywhere databases. This client functionality is accessed through stored functions and procedures.

In addition, the term web services also refers to applications that use the built-in web server to handle HTTP requests from clients. These applications generally function like traditional database-backed web applications, but can be more compact and are easier to write as the data and the entire application can reside within a database. In this type of application, the web service typically returns documents in HTML format. The GET, HEAD, and POST methods are supported.

The collection of web services within your database together define the available URLs. Each service provides a set of web pages. Typically, the content of these pages is generated by procedures that you write and store in your database, although they can be a single statement or, optionally, allow users to execute statements of their own. These web services become available when you start the database server with options that enable it to listen for HTTP requests.

Since the HTTP server that handles web service requests is embedded in the database, performance is good. Applications that use web services are easily deployed, since no additional components are needed, beyond the database and database server.
Quick start to web services

The following procedure describes how to create a new database, start a SQL Anywhere database with the HTTP server enabled, and access this database using any popular web browser.

To create and access a simple HTML web service

1. At a command prompt, run the following command to start a personal web server. Replace samples-dir with the actual location of the sample database. The -xs http(port=80) option tells the database server to listen for HTTP requests. If you already have a web server running on port 80, use another port number such as 8080 for this demonstration.

   `dbeng11 -xs http(port=80) samples-dir\demo.db`

   Many properties of the HTTP communication link are controlled by parameters to the -xs option. See “-xs server option” [SQL Anywhere Server - Database Administration].

2. Start the database server with the appropriate -xs option parameters and use the CREATE SERVICE statement to create web services to respond to incoming requests.

   Start Interactive SQL. Connect to the SQL Anywhere sample database as the DBA. Execute the following statement.

   ```
   CREATE SERVICE HTMLtable
   TYPE 'HTML'
   AUTHORIZATION OFF
   USER DBA
   AS SELECT * FROM Customers;
   ```

   This statement creates a web service named HTMLtable. This simple web service returns the results of the statement SELECT * FROM Customers, automatically converting the output into HTML format. Because authorization is off, no permissions are required to access the table from a web browser. See “Creating web services” on page 856 and “CREATE SERVICE statement” [SQL Anywhere Server - SQL Reference].

3. Start a web browser.

4. Browse to the URL http://localhost:80/demo/HTMLtable. Use the port number you specified when starting the database server.

   Your web browser shows you the body of the HTML document returned by the database server. By default, the result set is formatted into an HTML table.

To create and access a simple XML web service

1. At a command prompt, run the following command to start a personal web server. Replace samples-dir with the actual location of the sample database. The -xs http(port=80) option tells the database server to listen for HTTP requests. If you already have a web server running on port 80, use another port number such as 8080 for this demonstration.

   `dbeng11 -xs http(port=80) samples-dir\demo.db`
Many properties of the HTTP communication link are controlled by parameters to the -xs option. See “-xs server option” [SQL Anywhere Server - Database Administration].

2. Start the database server with the appropriate -xs option parameters and use the CREATE SERVICE statement to create web services to respond to incoming requests.

Start Interactive SQL. Connect to the SQL Anywhere sample database as the DBA. Execute the following statement:

```
CREATE SERVICE XMLtable
    TYPE 'XML'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM Customers;
```

This statement creates a web service named XMLtable. This simple web service returns the results of the statement SELECT * FROM Customers, automatically converting the output into XML format. Because authorization is off, no permissions are required to access the table from a web browser. See “Creating web services” on page 856 and “CREATE SERVICE statement” [SQL Anywhere Server - SQL Reference].

3. Start a web browser.

4. Browse to the URL http://localhost:80/demo/XMLtable. Use the port number you specified when starting the database server.
   - localhost:80 Defines the web host name and port number to use.
   - demo Defines the database name to use. You are using demo.db.
   - XMLtable Defines the service name to use.

   Your web browser shows you the body of the XML document returned by the database server. As no formatting information has been included, you see the raw XML, including tags and attributes.

5. You can also access the XMLtable service from common programming languages. For example, the following short C# program uses the XMLtable web service:

```csharp
using System.Xml;

static void Main(string[] args)
{
    XmlTextReader reader =
        new XmlTextReader( "http://localhost:80/demo/XMLtable" );

    while( reader.Read() )
    {
        switch( reader.NodeType )
        {
            case XmlNodeType.Element:
                if( reader.Name == "row" )
                {
                    Console.Write(reader.GetAttribute("ID")+" ");
                    Console.WriteLine(reader.GetAttribute("Surname"));
                }
                break;
        }
    }
    reader.Close();
}
```
6. In addition, you can access the same web service from Python, as in the following example:

```python
import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_id = attrs.getValue( 'ID' )
            table_name = attrs.getValue( 'Surname' )
            print '%s %s' % ( table_id, table_name )

parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse( 'http://localhost:80/demo/XMLtable' )
```

Save this code in a file called `DocHandler.py`. To run the application, enter a command like the following:

```bash
python DocHandler.py
```

**To create and access a simple JSON web service**

1. At a command prompt, run the following command to start a personal web server. Replace `samples-dir` with the actual location of the sample database. The `-xs http(port=80)` option tells the database server to listen for HTTP requests. If you already have a web server running on port 80, use another port number such as 8080 for this demonstration.

   ```bash
dbeng11 -xs http(port=80) samples-dir\demo.db
```

   Many properties of the HTTP communication link are controlled by parameters to the `-xs` option. See “-xs server option” [SQL Anywhere Server - Database Administration].

2. Start the database server with the appropriate `-xs` option parameters, and use the CREATE SERVICE statement to create web services to respond to incoming requests.

Start Interactive SQL. Connect to the SQL Anywhere sample database as the DBA. Execute the following statement:

```sql
CREATE SERVICE JSONtable
    TYPE 'JSON'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM Customers;
```

This statement creates a web service named JSONtable. This simple web service returns the results of the statement `SELECT * FROM Customers`, automatically converting the output into JavaScript Object Notation format. Because authorization is off, no permissions are required to access the table from a web browser. See “Creating web services” on page 856 and “CREATE SERVICE statement” [SQL Anywhere Server - SQL Reference].

3. Start a web browser.

4. Browse to the URL `http://localhost:80/demo/JSONtable`. Use the port number you specified when starting the database server.

   - **localhost:80** Defines the web host name and port number to use.
   - **demo** Defines the database name to use. You are using `demo.db`. 

---

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
- **JSONtable** Defines the service name to use.

Your web browser should permit you to save the JSON response document returned by the database server. Save the response to a file.

5. If you use a text editor to view the file containing the response, you will see the following array notation for the result set.

```
[
  {
    "ID": 101,
    "Surname": "Devlin",
    "GivenName": "Michaels",
    "Street": "114 Pioneer Avenue",
    "City": "Kingston",
    "State": "NJ",
    "Country": "USA",
    "PostalCode": "07070",
    "Phone": "2015558966",
    "CompanyName": "The Power Group"
  },
  {
    "ID": 102,
    "Surname": "Reiser",
    "GivenName": "Beth",
    "Street": "33 Whippany Road",
    "City": "Rockwood",
    "State": "NY",
    "Country": "USA",
    "PostalCode": "10154",
    "Phone": "2125558725",
    "CompanyName": "AMF Corp."
  },
  {
    "ID": 665,
    "Surname": "Thompson",
    "GivenName": "William",
    "Street": "19 Washington Street",
    "City": "Bancroft",
    "State": "NY",
    "Country": "USA",
    "PostalCode": "11700",
    "Phone": "5165552549",
    "CompanyName": "The Apple Farm"
  }
]
```

**Other resources for getting started**

Samples are included in the `samples-dir\SQLAnywhere\HTTP` directory.

See also

- “HTML_DECODE function [Miscellaneous]” [SQL Anywhere Server - SQL Reference]
- “HTML_ENCODE function [Miscellaneous]” [SQL Anywhere Server - SQL Reference]
- “HTTP_DECODE function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “HTTP_ENCODE function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “HTTP_HEADER function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “HTTP_VARIABLE function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “NEXT_HTTP_HEADER function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “NEXT_HTTP_VARIABLE function [HTTP]” [SQL Anywhere Server - SQL Reference]
- “NEXT_SOAP_HEADER function [SOAP]” [SQL Anywhere Server - SQL Reference]
- “SOAP_HEADER function [SOAP]” [SQL Anywhere Server - SQL Reference]
- “sa_http_header_info 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]
Creating web services

Web services, created and stored in databases, define which URLs are valid and what they do. A single database can define multiple web services. It is possible to define web services in different databases so that they appear to be part of a single web site.

The following statements permit you to create, alter, and delete web services:

- CREATE SERVICE
- ALTER SERVICE
- DROP SERVICE
- COMMENT ON SERVICE

The general syntax of the CREATE SERVICE statement is as follows:

```
CREATE SERVICE service-name TYPE 'service-type' [ attributes ] [ AS statement ]
```

Service names

Since service names form part of the URL used to access them, they are flexible in terms of what characters they can contain. In addition to the standard alpha-numeric characters, the following characters are permitted:

- _ . ! * '( )

In addition, service names other than those used in naming DISH services can contain a slash, "/", but some restrictions apply because this character is a standard URL delimiter and affects how SQL Anywhere interprets your URLs. It cannot be the first character of a service name. In addition, service names cannot contain two consecutive slashes.

The characters permitted in service names are also permitted in GROUP names, which apply to DISH services only.

Service types

The following service types are supported:

- 'SOAP'  The result set is returned as a SOAP response. The format of the data is determined by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a simple HTTP request.
- 'DISH'   A DISH service (Determine SOAP Handler) acts as a proxy for those SOAP services identified by the GROUP clause, and generates a WSDL (Web Services Description Language) document for each of these SOAP services.
- 'HTML'   The result set of a statement or procedure is automatically formatted into an HTML document that contains a table.
- 'XML'    The result set is returned as XML. If the result set is already XML, no additional formatting is applied. If it is not already XML, it is automatically formatted as XML. The effect is similar to that of using the FOR XML RAW clause in a SELECT statement.
● **'JSON'**  The result set is returned in JavaScript Object Notation (JSON). JSON is more compact than XML and has a similar structure. For more information about JSON, visit [http://www.json.org](http://www.json.org).

● **'RAW'**  The result set is sent to the client without any further formatting. You can produce formatted documents by generating the required tags explicitly within your procedure.

Of all the service types, RAW gives you the most control over the output. However, it does require that you do more work as you must explicitly output all the necessary tags. The output of XML services can be adjusted by applying the FOR XML clause to the service's statement. The output of SOAP services can be adjusted using the FORMAT attribute of the CREATE or ALTER SERVICE statement. See “CREATE SERVICE statement” [*SQL Anywhere Server - SQL Reference*].

**Statements**

The statement is the command, usually a stored procedure, that is called when someone accesses the service. If you define a statement, this is the only statement that can be run through this service. The statement is mandatory for SOAP services, and ignored for DISH services. The default is NULL, which means no statement.

You can create services that do not include statements. The statement is taken from the URL. Services configured in this way can be useful when you are testing a service, or want a general way of accessing information. To do so, either omit the statement entirely or use the phrase AS NULL in place of the statement.

Services without statements are a serious security risk because they permit web clients to execute arbitrary commands. When creating such services, you must enable authorization, which forces all clients to provide a valid user name and password. Even so, only services that define statements should be run in a production system.

**Attributes**

In general, all attributes are optional. However, some are interdependent. The following attributes are available:

- **AUTHORIZATION**  This attribute controls which users can use the service. The default setting is ON. Authorization must be ON if no statement is provided. In addition, the authorization setting affects how user names, defined by the USER attribute, are interpreted.

- **SECURE**  When set to ON, only secure connections are permitted. All connections received on the HTTP port are automatically redirected to the HTTPS port. The default is OFF, which enables both HTTP and HTTPS requests, provided these ports are enabled using the appropriate options when the database server is started. See “-xs server option” [*SQL Anywhere Server - Database Administration*].

- **USER**  The USER clause controls which database user accounts can be used to process service requests. However, the interpretation of this setting depends on whether authorization is ON or OFF.

When authorization is set to ON, all clients must provide a valid user name and password when they connect. When authorization is ON, the USER option can be NULL, a database user name, or the name of a database group. If it is NULL, any database user can connect and make requests. Requests are run using the account and permissions of that user. If a group name is specified, only those users who belong to the group can run requests. All other database users are denied permission to use the service.

If authorization is OFF, a statement must be provided. In addition, a user name must be provided. All requests are run using that user's account and permissions. So, if the server is connected to a public
network, the permissions of the named user account should be minimal to limit the damage that could be caused through malicious use.

- **GROUP**  
The GROUP clause, which applies to DISH services only, determines which SOAP services are exposed by the DISH service. Only SOAP services whose names begin with the name of the group name of a DISH service are exposed by that DISH service. So, the group name is a common prefix among the exposed SOAP services. For example, specifying GROUP xyz exposes only SOAP services xyz/aaaa, xyz/bbbb, or xyzcccc, but does not expose abc/aaaaa or xyzaaaaa. If no group name is specified, the DISH service exposes all the SOAP services in the database. The same characters are permitted in group names as in service names.

SOAP services can be exposed by more than one DISH service. In particular, this feature permits a single SOAP service to supply data in multiple formats. The service type, unless specified in a SOAP service, is inherited from the DISH service. So, you can create a SOAP service that declares no format type, then include it in multiple DISH services, each of which specifies a different format.

- **FORMAT**  
The FORMAT clause, which applies to DISH and SOAP services only, controls the output format of the SOAP or DISH response. Output formats compatible with various types of SOAP clients, such as .NET or JAX-WS, are available. If the format of a SOAP service is not specified, the format is inherited from the service's DISH service declaration. If the DISH service also does not declare a format, it defaults to DNET, which is compatible with .NET clients. A SOAP service that does not declare a format can be used with different types of SOAP clients by defining multiple DISH services, each having a different FORMAT type.

- **URL [PATH]**  
The URL or URL PATH clause controls the interpretation of URLs and applies to XML, HTML, and RAW service types only. In particular, it determines whether URL paths are accepted and, if so, how they are processed. If the service name ends with the character "/", URL must be set to OFF. See “CREATE SERVICE statement” [SQL Anywhere Server - SQL Reference].
Starting a database server that listens for web requests

When you want a database server to listen for web service requests over HTTP or HTTPS, you must specify the types of web requests it is to listen on the command line when you start the server. By default, database servers do not listen for web service requests, leaving no way for clients to access any services that may be defined in your database.

You can also specify various properties of an HTTP or HTTPS service on the command line, such as on which port they are to listen.

You must also create web services within the database. See “Creating web services” on page 856.

You use the -xs option to enable protocols. The two available web service protocols are HTTP and HTTPS. Optional parameters, placed within parentheses after the protocol name, let you customize access to each type of web service.

The general syntax of the option is as follows:

```
-xs { protocol [ (option=value; ...) ], ... }
```

### Starting multiple web servers

If you want to start multiple web servers at the same time, then you must change the port for additional web servers since they all have the same default port.

### Protocols

The following web service protocol values are available:

- **http**  
  Listen for HTTP connections.

- **https**  
  Listen for HTTPS connections. HTTPS connections using SSL version 3.0 and TLS version 1.0 are supported.

- **none**  
  Do not listen for web service requests. This is the default setting.

### Options

The following are some of the options that are available:

- **FIPS**  
  Specify $FIPS=Y$ to listen for HTTPS FIPS connections.

- **ServerPort [PORT]**  
  The port on which to listen for web requests. By default, SQL Anywhere listens on port 80 for HTTP requests and on port 443 for secure HTTP (HTTPS) requests. The default port for FIPS-approved HTTPS connections is the same as for HTTPS.

For example, if you already have a web server running on port 80, you could use the following options to start a database server that listens for web requests on port 8080:

```
dbeng11 mywebapp.db -xs http(port=8080)
```
As another example, the following command starts a secure web server using the sample identity file included with SQL Anywhere (you must have installed RSA or FIPS-approved RSA encryption to have this file). It should be entered on a single line.

dbeng11 -xs https(identity=rsaserver.id; identity_password=test)

**Caution**
The sample identity file is intended for use only during testing and development. It provides no protection because it is a standard part of SQL Anywhere. Replace it with your own certificate before deploying your application.

- **DatabaseName [DBN]** Specifies the name of a database to use when processing web requests, or uses the REQUIRED or AUTO keyword to specify whether database names are required as part of the URL.

  If this parameter is set to REQUIRED, the URL must specify the database name.

  If this parameter is set to AUTO, the URL may specify a database name, but does not need to do so. If the URL contains no database name, the default database on the server is used to process web requests.

  If this parameter is set to the name of a database, that database is used to process all web requests. The URL must not contain a database name.

- **LocalOnly [LOCAL]** When set to YES, this parameter causes a network database server to reject all connections from clients running on different computers. This option has no effect on personal database servers, which never accept web service requests from other computers. The default value is NO, which means accept requests from clients no matter where they are located.

- **LogFile [LOG]** The name of the file to which the database server is to write information about web service requests.

- **LogFormat [LF]** Controls the format of messages written to the log file and which fields appear in them. If they appear in the string, current values are substituted for the codes, such as @T, when each message is written.

  The default value is @T - @W - @I - @P - "@M @U @V" - @R - @L - @E, which produces messages such as the following:

```plaintext
06/15 01:30:08.114 - 0.686 - 127.0.0.1 - 80
- "GET /web/ShowTable HTTP/1.1" - 200 OK - 55133 -
```

  The format of the log file is compatible with Apache, so the same tools can be used to analyze it.

  For more information about field codes, see “LogFormat protocol option [LF]” [SQL Anywhere Server - Database Administration].

- **LogOptions [LOPT]** Allows you to specify keyword and error numbers that control which messages, or types of messages, are written to the log file. See “LogOptions protocol option [LOPT]” [SQL Anywhere Server - Database Administration].

  For a complete list of the available options and details about them, see “Network protocol options” [SQL Anywhere Server - Database Administration].
Understanding how URLs are interpreted

Universal Resource Locators, or URLs, identify documents, such as HTML pages, available from SOAP or HTTP web services. The URLs used in SQL Anywhere follow the patterns familiar to you from browsing the web. Users browsing through a database server need not be aware that their requests are not being handled by a traditional standalone web server.

Although standard in format, SQL Anywhere database servers interpret URLs differently than standard web servers. The options you specify when you start the database server also affect their interpretation.

The general syntax of the URL is as follows:

{ http | https }://[ user:password@ ]host[ :port ][ /dbname ]/service-name[ path | ?searchpart ]

The following is an example URL: http://localhost:80/demo/XMLtable.

User and password

When a web service requires authentication, the user name and password can be passed directly as part of the URL by separating them with a colon and prepending them to the host name, much like an email address.

Host and port

Like all standard HTTP requests, the start of the URL contains the host name or IP number and, optionally, a port number. The IP address or host name, and port, should be the one on which your server is listening. The IP address is the address of a network card in the computer running SQL Anywhere. The port number will be the port number you specified using the -xs option when you started the database server. If you did not specify a port number, the default port number for that type of service is used. For example, the server listens by default on port 80 for HTTP requests. See “-xs server option” [SQL Anywhere Server - Database Administration].

Database name

The next token, between the slashes, is usually the name of a database. This database must be running on the server and must contain web services.

The default database is used if no database name appears in the URL and the database name was not specified using the DBN connection parameter to the -xs server option.

The database name can be omitted only if the database server is running only one database, or if the database name was specified using the DBN connection parameter to the -xs option.

Service name

The next portion of the URL is the service name. This service must exist in the specified database. The service name can extend beyond the next slash character because web service names can contain slash characters. SQL Anywhere matches the remainder of the URL with the defined services.

If the URL provides no service name, then the database server looks for a service named root. If the named service, or the root service, is not defined, then the server returns a 404 Not Found error.
Parameters

Depending on the type of the target service, parameters can be supplied in different ways. Parameters to HTML, XML, and RAW services can be passed in any of the following ways:

- appended to the URL using slashes
- supplied as an explicit URL parameters list
- supplied as POST data in a POST request

Parameters to SOAP services must be included as part of a standard SOAP request. Values supplied in other ways are ignored.

URL path

To access parameter values, parameters must be given names. These host variable names, prefixed with a colon (:), can be included in the statement that forms part of the web service definition.

For example, suppose you define the following stored procedure:

```sql
CREATE PROCEDURE Display (IN ident INT )
BEGIN
  SELECT ID, GivenName, Surname FROM Customers
  WHERE ID = ident;
END;
```

A statement that calls the stored procedure requires a customer identification number. Define the service as follows:

```sql
CREATE SERVICE DisplayCustomer
  TYPE 'HTML'
  URL PATH ELEMENTS
  AUTHORIZATION OFF
  USER DBA
  AS CALL Display( :url1 );
```

An example of a URL for this is: `http://localhost:80/demo/DisplayCustomer/105`.

The parameter 105 is passed as `url1` to the service. The clause URL PATH ELEMENTS indicates that parameters separated by slashes should be passed as parameters `url1`, `url2`, `url3`, and so on. Up to 10 parameters can be passed in this way.

Since there is only one parameter to the Display procedure, the service could have been defined like this:

```sql
CREATE SERVICE DisplayCustomer
  TYPE 'HTML'
  URL PATH ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL Display( :url );
```

In this case, the parameter 105 would be passed as `url` to the service. The clause URL PATH ON indicates that everything after the service name should be passed as a single parameter called `url`. So in the following URL, the string 105/106 would be passed as `url` (and a SQL error would result since the Display stored procedure requires an integer value).

`http://localhost:80/demo/DisplayCustomer/105/106`
For more information about variables, see “Working with variables” on page 917.

Parameters can also be accessed using the HTTP_VARIABLE function. See “HTTP_VARIABLE function [HTTP]” [SQL Anywhere Server - SQL Reference].

**URL searchpart**

Another method for passing parameters is through the URL searchpart mechanism. A URL searchpart consists of a question mark (?) followed by name=value pairs separated by ampersands (&). The searchpart is appended to the end of a URL. The following example shows the general format:

```
http://server/path/document?name1=value1&name2=value2
```

GET requests are formatted in this manner. If present, the named variables are defined and assigned the corresponding values.

For example, a statement that calls the stored procedure ShowSalesOrderDetail requires both a customer identification number and a product identification number:

```sql
CREATE SERVICE ShowSalesOrderDetail
  TYPE 'HTML'
  URL PATH OFF
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

An example of a URL for this is: http://localhost:80/demo/ShowSalesOrderDetail?customer_id=101&product_id=300.

If you have URL PATH set to ON or ELEMENTS, additional variables are defined. However, the two are usually otherwise independent. You can allow variables to be used in requested URLs by setting URL PATH to ON or ELEMENTS. The following example illustrates how the two can be mixed:

```sql
CREATE SERVICE ShowSalesOrderDetail2
  TYPE 'HTML'
  URL PATH ON
  AUTHORIZATION OFF
  USER DBA
  AS CALL ShowSalesOrderDetail( :customer_id, :url );
```

In the following example, both searchpart and URL path are used. The value 300 is assigned to url and 101 is assigned to customer_id.

http://localhost:80/demo/ShowSalesOrderDetail2/300?customer_id=101

This can also be expressed using searchpart only in the following manner.

http://localhost:80/demo/ShowSalesOrderDetail2/?customer_id=101&url=300

This then leads to the question of what happens when both are specified for the same variable. In the following example, first 300 and then 302 are assigned to url in sequence and it is the last assignment that takes precedence.


For more information about variables, see “Working with variables” on page 917.
Parameters can also be accessed using the HTTP_VARIABLE function. See “HTTP_VARIABLE function [HTTP]” [SQL Anywhere Server - SQL Reference].
Creating SOAP and DISH web services

SOAP and DISH web services are the means by which you create standard SOAP web services that can be accessed by standard SOAP clients, such as those written with Microsoft .NET or JAX-WS.

SOAP services

SOAP services are the mechanism for constructing web services in SQL Anywhere that accept and process standard SOAP requests.

To declare a SOAP service, specify that the service is to be of type SOAP. The body of a standard SOAP request is SOAP envelope, meaning an XML document with a specific format. SQL Anywhere parses and processes these requests using the procedures you provide. The response is automatically formatted in the form of a standard SOAP response, which is also a SOAP envelope, and returned to the client.

The syntax of the statement used to create SOAP services is as follows:

```sql
CREATE SERVICE service-name
TYPE 'SOAP'
[ FORMAT { 'DNET' | 'CONCRETE' | 'XML' | NULL } ]
[ common-attributes ]
AS statement
```

DISH services

DISH services act as proxies for groups of SOAP services. In addition, they automatically construct WSDL (Web Services Description Language) documents for their clients that describe the SOAP services that they currently expose.

When you create a DISH service, the name given in the GROUP clause determines which SOAP services the DISH service exposes. Every SOAP service whose name is prefixed with the name of the DISH service is exposed. For example, specifying GROUP xyz exposes SOAP services xyz/aaaa, xyz/bbbb, or xyz/ccccc. It does not expose SOAP services named abc/aaaa or xyzaaaa. SOAP services can be exposed by more than one DISH service. If no group name is specified, the DISH service exposes all the SOAP services in the database. The same characters are permitted in DISH group names as in SOAP service names.

The syntax of the statement used to create DISH services is as follows:

```sql
CREATE SERVICE service-name
TYPE 'DISH'
[ GROUP { group-name | NULL } ]
[ FORMAT { 'DNET' | 'CONCRETE' | 'XML' | NULL } ]
[ common-attributes ]
```

SOAP and DISH service formats

The FORMAT clause of the CREATE SERVICE statement customizes the SOAP service data payload to best suit the various types of SOAP clients, such as .NET and JAX-WS. The FORMAT clause affects the content of the WSDL document returned by a DISH service and the format of data payloads returned in SOAP responses.

The default format, DNET, is a native format for use with .NET SOAP client applications, which expect a .NET DataSet format.
The **CONCRETE** format is for use with clients such as JAX-WS and .NET that automatically generate interfaces based on the format of the returned data structures. When you specify this format, the WSDL document returned by SQL Anywhere exposes a SimpleDataset element that describes a result set in concrete terms. This element is a containment hierarchy of a rowset composed of an array of rows, each containing an array of column elements.

The **XML** format is for use with SOAP clients that accept the SOAP response as one large string, and use an XML parser to locate and extract the required elements and values. This format is generally the most portable between different types of SOAP clients.

If the format of a SOAP service is not specified, the format is inherited from the service's DISH service declaration. If the DISH service also does not declare a format, it defaults to DNET, which is compatible with .NET clients. A SOAP service that does not declare a format can be used with different types of SOAP clients by defining multiple DISH services, each having a different FORMAT type.

### Creating homogeneous DISH services

SOAP services need not specify a format type—you can set the format type to NULL. In this case, the format is inherited from the DISH services that act as proxies for them. More than one DISH service can act as a proxy for each SOAP service, and these DISH services need not be of the same type. These facts mean that it is possible to use a single SOAP service with different types of SOAP clients, such as .NET and JAX-WS, by using multiple DISH services, each of a different type. Such DISH services are said to be **homogeneous** because they expose the same data payloads for the same SOAP services, but in different formats.

For example, consider the following two SOAP services, neither of which specifies a format:

```sql
CREATE SERVICE "abc/hello"
TYPE 'SOAP'
AS CALL hello(:student);

CREATE SERVICE "abc/goodbye"
TYPE 'SOAP'
AS CALL goodbye(:student);
```

Since neither of these services includes a FORMAT clause, the format, by default, is NULL; it is inherited from the DISH service that is acting as a proxy. Now, consider the following two DISH services:

```sql
CREATE SERVICE "abc_xml"
TYPE 'DISH'
GROUP "abc"
FORMAT 'XML';

CREATE SERVICE "abc_concrete"
TYPE 'DISH'
GROUP "abc"
FORMAT 'CONCRETE';
```

Since both DISH services specify the same group name abc, they act as proxies for the same SOAP services, namely all SOAP services whose names have the prefix "abc/".

However, when either of the two SOAP services is accessed through the abc_xml DISH service, the SOAP service inherits the XML format; when accessed through the abc_concrete SOAP service, the SOAP service inherits the CONCRETE format.
Homogeneous DISH services provide a means of avoiding duplicate services whenever you want to give different types of SOAP clients access to the SOAP web services you create.
Tutorial: Accessing web services from Microsoft .NET

The following tutorial demonstrates how to access web services from Microsoft .NET using Visual C#.

**To create SOAP and DISH services**

1. At a command prompt, run the following command to start a personal web server. Replace `samples-dir` with the actual location of the sample database. The `-xs http(port=80)` option tells the database server to accept HTTP requests on port 80. If you already have a web server running on port 80, use another port number such as 8080 for this tutorial and use 8080 instead of 80 in all port references.

   ```
   dbeng11 -xs http(port=80) samples-dir\demo.db
   ```

2. Start Interactive SQL. Connect to the SQL Anywhere sample database as the DBA. Execute the following statements:
   a. Define a SOAP service that lists the Employees table.

   ```
   CREATE SERVICE "SASoapTest/EmployeeList"
   TYPE 'SOAP'
   AUTHORIZATION OFF
   SECURE OFF
   USER DBA
   AS SELECT * FROM Employees;
   ```

   Because authorization has been turned off, anyone can use this service without supplying a user name and password. The commands run under user DBA. This arrangement is simple, but insecure.

   b. Create a DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

   ```
   CREATE SERVICE "SASoapTest_DNET"
   TYPE 'DISH'
   GROUP "SASoapTest"
   FORMAT 'DNET'
   AUTHORIZATION OFF
   SECURE OFF
   USER DBA;
   ```

   The SOAP and DISH service must be of format DNET. In this example, the FORMAT clause was omitted when the SOAP service was created. As a result, the SOAP service inherits the DNET format from the DISH service.

3. Start Microsoft Visual Studio. Note that this example uses functions from the .NET Framework 2.0.
   a. Create a new Windows application project using Visual C#.

   An empty form appears.

   b. From the **Project** menu, choose **Add Web Reference**.

   c. In the **URL** field of the **Add Web Reference** page, enter the following URL: `http://localhost:80/demo/SASoapTest_DNET`.

   d. Click **Go**.
You are presented with a list of the methods available for SASoapTest_DNET. You should see the EmployeeList method.

e. Click **Add Reference** to finish.

   The **Solution Explorer** window shows the new web reference.

f. From the Visual Studio **Toolbox**, add a listbox and a button to the form as shown in the following diagram.

![Form1](image)

   Click **Employee List**.

h. Double-click **Employee List** and add the following code for the button click event.

   ```csharp
   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.GetName(i);
         try
         {
            string value = dr.GetString(i);
            listBox1.Items.Add(columnName + "=" + value);
         }
         catch (InvalidCastException)
         {
            listBox1.Items.Add(columnName + "=(null)");
         }
      }
   }
   ```
i. Build and run the program.

The listbox will display the EmployeeList result set as column name=value pairs.

A try/catch block is included to handle NULL column values such as those found in the TerminationDate column of the Employees table.
Tutorial: Accessing web services from JAX-WS

The following tutorial demonstrates how to access web services using the Java API for XML Web Services (JAX-WS).

Before you begin, you will need the JAX-WS tools available from Sun. If you do not have this package installed on your system, you must download the JAX-WS tools and install them. To download the JAX-WS tools, visit http://java.sun.com/webservices/. Click the link for JAX-WS. This should take you to the Java API for XML Web Services page. Click the Download Now link. After you have downloaded the software package, install it on your system.

This example was developed with JAX-WS 2.1.3 for Windows.

SQL Anywhere SOAP web services that are accessed from JAX-WS should be declared to be of format CONCRETE.

To create SOAP and DISH services

1. At a command prompt, run the following command to start a personal web server. Replace samples-dir with the actual location of the sample database. The -xs http(port=80) option tells the database server to accept HTTP requests on port 80. If you already have a web server running on port 80, use another port number such as 8080 for this tutorial and use 8080 instead of 80 in all port references.

   dbeng11 -xs http(port=80) samples-dir\demo.db

2. Start Interactive SQL and connect to the SQL Anywhere sample database as the DBA. Execute the following statements:

   a. Define a stored procedure that lists some columns of the Employees table.

      ```
      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;
      ```

   b. Define a SOAP service that calls this stored procedure.

      ```
      CREATE SERVICE "WS/EmployeeList" 
      TYPE 'SOAP' 
      FORMAT 'CONCRETE' EXPLICIT OFF 
      DATATYPE OUT 
      AUTHORIZATION OFF 
      SECURE OFF 
      USER DBA 
      AS CALL ListEmployees();
      ```

      The EXPLICIT clause can only be used with a SOAP or DISH service of type CONCRETE. In this example, EXPLICIT OFF indicates that the corresponding DISH service should generate XML Schema that describes the generic SimpleDataset object. This option only affects the WSDL.
document that is generated. Later on, we will look at an example that uses EXPLICIT ON. See “Tutorial: Using data types with JAX-WS” on page 892.

DATATYPE OUT indicates that explicit data type information is generated in the XML result set response. If DATATYPE OFF had been specified, then all data would be typed as string. This option does not affect the WSDL document that is generated.

Because authorization has been turned off, anyone can use this service without supplying a user name and password. The commands run under user DBA. This arrangement is simple, but not secure.

c. Create a DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

```
CREATE SERVICE "WSDish"
TYPE 'DISH'
FORMAT 'CONCRETE'
GROUP "WS"
AUTHORIZATION OFF
SECURE OFF
USER DBA;
```

The SOAP and DISH service must both be format CONCRETE. Since the EmployeeList service is in the WS group, a GROUP clause is included.

3. Take a look at the WSDL that the DISH service automatically creates. To do so, open a web browser and browse to the following URL: http://localhost:80/demo/WSDish. The DISH service automatically generates a WSDL document that appears in the browser window.

In particular, observe the SimpleDataset object that is exposed because the format of this service is CONCRETE and EXPLICIT is OFF. In a later step, the wsimport application uses this information to generate a SOAP 1.1 client interface for this service.

```
<s:complexType name="SimpleDataset">
  <s:sequence>
    <s:element name="rowset">
      <s:complexType>
        <s:sequence>
          <s:element name="row" minOccurs="0" maxOccurs="unbounded">
            <s:complexType>
              <s:sequence>
                <s:any minOccurs="0" maxOccurs="unbounded" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:sequence>
</s:complexType>
```

To generate a JAX-WS interface for these web services

1. In this example, the Java API for XML Web Services (JAX-WS) and the Sun Java 1.6.0 JDK are installed on drive C:. Set your PATH environment variable so that it includes the JAX-WS binaries and the JDK binaries. The binaries are located in the following directories:

```
c:\Sun\jaxws-ri\bin
c:\Sun\SDK\jdk\bin
```
2. At a command prompt, set your CLASSPATH environment variable.

   SET classpath=.;C:\Sun\jaxws-ri\lib\jaxb-api.jar
   ;C:\Sun\jaxws-ri\lib\jaxws-rt.jar

3. The next step is to generate the interface needed to call the web service.

   At the same command prompt, create a new project directory and make this directory your current directory. Execute the following command in this directory.

   wsimport -keep -Xendorsed "http://localhost:80/demo/WSDish"

   The wsimport tool retrieves the WSDL document from the given URL, generates the Java files that define the interface for it, and then compiles the Java files.

   The keep option tells wsimport not to delete the .java files. Without this option, these files are deleted after the corresponding .class files have been generated. Saving these files makes it easier to examine the makeup of the interface.

   The Xendorsed option allows you to use the JAX-WS 2.1 API with JDK6.

   Once this command completes, you should have a new subdirectory structure named localhost\demo \ws in your current directory that contains the following Java files, along with the compiled .class versions of each Java file.

   EmployeeList.java
   EmployeeListResponse.java
   FaultMessage.java
   ObjectFactory.java
   package-info.java
   SimpleDataset.java
   WSDish.java
   WSDishSoapPort.java

To use the generated JAX-WS interface

1. Save the following Java source code into SASoapDemo.java. Make sure that this file is located in the same directory containing the localhost subdirectory that was generated by the wsimport tool.

   // 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 localhost.demo.ws.*;

   public class SASoapDemo
   {
      @WebServiceRef( wsdlLocation= "http://localhost:8080/demo/WSDish" )
      public static void main( String[] args )
      {
         try {
            WSDish service = new WSDish();

            Holder<SimpleDataset> response = new Holder<SimpleDataset>();
            Holder<Integer> sqlcode = new Holder<Integer>();

            WSDishSoapPort port = service.getWSDishSoap();
// This is the SOAP service call to EmployeeList
port.employeeList( response, sqlcode );

localhost.demo.ws.SimpleDataset result = response.value;
SimpleDataset.Rowset rowset = result.getRowset();

List<SimpleDataset.Rowset.Row> rows = rowset.getRow();

for ( int i = 0; i < rows.size(); i++ ) {
    SimpleDataset.Rowset.Row row = rows.get( i );
    List<Object> cols = row.getAny();
    System.out.println( "Number of columns=" + cols.size() );

    for ( int j = 0; j < cols.size(); j++ ) {
        // Column data is contained as a SOAPElement
        Element col = (Element)cols.get(j);
        System.out.print( col.getLocalName() + "=");
        Node node = col.getFirstChild();
        if ( node == null ) {
            System.out.println( "(null)" );
        } else {
            System.out.println( node.getNodeValue() );
        }
    }
    System.out.println();
}

} catch (Exception x) {
    x.printStackTrace();
}

If you chose to start the web server using a different port number such as 8080, then you will have to alter the import localhost source line to something like the following:

import localhost._8080.demo.ws.*;

2. Compile SASoapDemo.java.

javac SASoapDemo.java

3. Run the compiled class file.

java SASoapDemo

When 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. Also included in the response is the sqlcode result from executing the query. An example of the response is shown next.

<tns:EmployeeListResponse>
    <tns:EmployeeListResult xsi:type='tns:SimpleDataset'>
        <tns:rowset>
            <tns:row>...</tns:row>
            ...
            ...
            <tns:row>...</tns:row>
    </tns:EmployeeListResult>
</tns:EmployeeListResponse>

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
Each row in the rowset is sent in a format similar to the following.

```xml
<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:date">1994-07-12-04:00</tns:StartDate>
  <tns:TerminationDate xsi:type="xsd:date">2008-04-18-04:00</tns:TerminationDate>
</tns:row>
```

Note that the column name and data type are included.

### Using a proxy

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.

The EmployeeList result set is displayed as (type)column name=value pairs by the SASoapDemo application. Several lines of output similar to the following should be generated.

The EmployeeList result set is displayed as column name=value pairs. Several lines of output similar to the following should be generated.

```
Number of columns=4
EmployeeID=102
Surname=Whitney
GivenName=Fran
StartDate=1984-08-28-04:00
```

```
Number of columns=4
EmployeeID=105
Surname=Cobb
GivenName=Matthew
StartDate=1985-01-01-05:00
```

```
Number of columns=4
EmployeeID=1740
Surname=Nielsen
GivenName=Robert
StartDate=1994-06-24-04:00
```

```
Number of columns=5
EmployeeID=1751
Surname=Ahmed
GivenName=Alex
StartDate=1994-07-12-04:00
TerminationDate=2008-04-18-04:00
```

Note that the TerminationDate column is only sent when its value is not NULL. For this example, the last row in the Employees table was altered such that a non-NUL termination date was set.

Also note that date values include an offset from UTC time. In the above sample data, the server is located in the North American Eastern time zone. This is 5 hours earlier than UTC time (-05:00) for dates when
daylight savings is not in effect and 4 hours earlier than UTC time (-04:00) for dates when daylight savings is in effect.
Using procedures that provide HTML documents

Generally, it is easiest to write a procedure that handles the requests sent to a particular service. Such a procedure should return a web page. Optionally the procedure can accept arguments, passed as part of the URL, to customize its output.

The following example, however, is much simpler. It demonstrates how simple a service can be. This web service simply returns the phrase "Hello world!".

```
CREATE SERVICE hello
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT 'Hello world!';
```

Start a database server with the -xs option to enable handling of web requests, and then request the URL http://localhost/hello from any web browser. The words Hello world! appear on an otherwise plain page.

HTML pages

The above page appears in your browser in plain text. This happens because the default HTTP Content-Type is text/plain. To create a more normal web page, formatted in HTML, you must do two things:

- Set the HTTP Content-Type header field to text/html so that the browsers expect HTML.
- Include HTML tags in the output.

You can write tags to the output in two ways. One way is to use the phrase TYPE 'HTML' in the CREATE SERVICE statement. Doing so instructs the SQL Anywhere database server to add HTML tags for you. This can work quite well if, for example, you are returning a table.

The other way is to use TYPE 'RAW' and write out all the necessary tags yourself. This second method provides the most control over the output. Note that specifying type RAW does not necessarily mean the output is not in HTML or XML format. It only tells SQL Anywhere that it can pass the return value directly to the client without adding tags itself.

The following procedure generates a fancier version of Hello world. For convenience, the body of the work is done in the following procedure, which formats the web page.

```
CREATE PROCEDURE hello_pretty_world ()
    RESULT (html_doc XML)
BEGIN
    CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
    SELECT HTML_DECODE(
        XMLCONCAT(
            '<!DOCTYPE HTML PUBLIC 
                "/W3C//DTD HTML 4.01//EN">',
            XMLELEMENT('HTML',
                XMLELEMENT('HEAD',
                    XMLELEMENT('TITLE', 'Hello Pretty World'),
                ),
                XMLELEMENT('BODY',
                    XMLELEMENT('H1', 'Hello Pretty World!'),
                ),
            )
        )
    );
```

The built-in procedure sa_set_http_header is used to set the HTTP header type so browsers interpret the result correctly. If you omit this statement, your browser displays all the HTML codes, rather than using them to format the document.
SQL Anywhere web services

XMLELEMENT('P',
'(If you see the tags in your browser, check that '
|| 'the Content-Type header is set to text/html.)'
)
)
)
)
);
END

The following statement creates a service that uses this procedure. The statement calls the above procedure,
which generates the Hello Pretty World web page.
CREATE SERVICE hello_pretty_world
TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL hello_pretty_world();

Once you have created the procedure and the service, you are ready to access the web page. Ensure that your
database server was started with the correct -xs option values, and then open the URL http://localhost/
hello_pretty_world in a web browser.
You see the results formatted in a simple HTML page, with the title Hello Pretty World. You can make the
web page as elaborate as you want by including more content, using more tags, using style sheets, or including
scripts that run in the browser. In all cases, you create the necessary services to handle the browser's requests.
For more information about built-in stored procedures, see “Alphabetical list of system procedures” [SQL
Anywhere Server - SQL Reference].
Root services
When no service name is included in a URL, SQL Anywhere looks for a web service named root. The role
of root pages is analogous to the role of index.html pages in many traditional web servers.
Root services are handy for creating home pages because they can handle URL requests that contain only
the address of your web site. They are also handy for handling URL paths that do not exist. For example,
the following procedure and service implement a simple web page that appears when you browse to the URL
http://localhost. It also handles the case where you browse to a page that does not exist.
CREATE PROCEDURE HomePage( IN url LONG VARCHAR )
RESULT (html_doc XML)
BEGIN
CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
IF url IS NULL THEN
SELECT HTML_DECODE(
XMLCONCAT(
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">',
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!'
)
)
)

878

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1


ELSE
    CALL dbo.sa_set_http_header('Status','404');
    SELECT '<H1>Status 404 - Document ' || url || ' not found</H1>'
END IF
END

Now create a service that uses this procedure:

    CREATE SERVICE root
    TYPE 'RAW'
    AUTHORIZATION OFF
    SECURE OFF
    URL PATH ON
    USER DBA
    AS CALL HomePage(:url);

You can access this web page by browsing to the URL http://localhost, as long as you do not specify that database names are mandatory when you start the database server. See “Starting a database server that listens for web requests” on page 859. By specifying URL PATH ON, you ensure that non-existent URL paths are directed to this service.

Examples

More extensive examples are included in the samples-dir\SQLAnywhere\HTTP directory.
Working with 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.

**DATATYPE { OFF | ON | IN | OUT }

- **OFF**  This is the default behavior when the DATATYPE option is not used. For DNET output format, SQL Anywhere data types are translated to and from XML Schema string types. For CONCRETE and XML formats, no data type information is emitted.

- **ON**  Data type information is emitted for both input parameters and result set responses. SQL Anywhere data types are translated to and from XML Schema data types.

- **IN**  Data type information is emitted for input parameters only.

- **OUT**  Data type information is emitted for result set responses only.

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:
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="0" maxOccurs="1" name="a_varchar" nillable="true" type="s:string" />
```

**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:sqlcode>0</tns:sqlcode>
    </tns:test_types_concrete_onResult>
  </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: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:sqlcode>0</tns:sqlcode>
    </tns:test_types_xml_onResult>
  </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:

```xml
<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>
    </tns:test_types_dnet_outResult xsi:type='sqlresultstream:SqlRowSet'>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>
```
### Mapping SQL Anywhere types to XML Schema types

<table>
<thead>
<tr>
<th>SQL Anywhere type</th>
<th>XML Schema type</th>
<th>XML Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>TEXT</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>NCHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>NTEXT</td>
<td>string</td>
<td>Hello World</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>string</td>
<td>12345678-1234-5678-9012-1234567890 12</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>string</td>
<td>12345678-1234-5678-9012-1234567890 12</td>
</tr>
<tr>
<td>XML</td>
<td>This is user defined. A parameter is assumed to be valid XML representing a complex type (for example, base64Binary, SOAP array, struct).</td>
<td><code>&lt;inputHexBinary xsi:type=&quot;xsd:hexBinary&quot;&gt; 414243 &lt;/inputHexBinary&gt;</code> (interpreted as 'ABC')</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
<td>-9223372036854775807</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>unsignedLong</td>
<td>18446744073709551615</td>
</tr>
<tr>
<td>BIT</td>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>VARBIT</td>
<td>string</td>
<td>11111111</td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>string</td>
<td>00000000000000000000000000000000000000</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>decimal</td>
<td>-1.756000</td>
</tr>
<tr>
<td>SQL Anywhere type</td>
<td>XML Schema type</td>
<td>XML Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
<td>.5555555555555555582</td>
</tr>
<tr>
<td>FLOAT</td>
<td>float</td>
<td>12.3456792831420898</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
<td>-2147483640</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>unsignedInt</td>
<td>4294967295</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>decimal</td>
<td>123456.123457</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
<td>3.25</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>-99</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>unsignedShort</td>
<td>33000</td>
</tr>
<tr>
<td>TINYINT</td>
<td>unsignedByte</td>
<td>128</td>
</tr>
<tr>
<td>MONEY</td>
<td>decimal</td>
<td>12345678.9900</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>decimal</td>
<td>12.3400</td>
</tr>
<tr>
<td>DATE</td>
<td>date</td>
<td>2006-11-21-05:00</td>
</tr>
<tr>
<td>DATETIME</td>
<td>dateTime</td>
<td>2006-05-21T09:00:00.000-08:00</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>dateTime</td>
<td>2007-01-15T09:00:00.000-08:00</td>
</tr>
<tr>
<td>TIME</td>
<td>time</td>
<td>14:14:48.980-05:00</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>dateTime</td>
<td>2007-01-12T21:02:14.420-06:00</td>
</tr>
<tr>
<td>BINARY</td>
<td>base64Binary</td>
<td>AAAAZg==</td>
</tr>
<tr>
<td>IMAGE</td>
<td>base64Binary</td>
<td>AAAAZg==</td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>base64Binary</td>
<td>AAAAZg==</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>base64Binary</td>
<td>AAAAZg==</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>XML Schema Type</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:integer</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>xsd:int</td>
<td>int</td>
</tr>
<tr>
<td>xsd:long</td>
<td>long</td>
</tr>
<tr>
<td>xsd:short</td>
<td>short</td>
</tr>
<tr>
<td>xsd:decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>xsd:float</td>
<td>float</td>
</tr>
<tr>
<td>xsd:double</td>
<td>double</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>xsd:byte</td>
<td>byte</td>
</tr>
<tr>
<td>xsd:QName</td>
<td>javax.xml.namespace.QName</td>
</tr>
<tr>
<td>xsd:dateTime</td>
<td>javax.xml.datatype.XMLGregorianCalendar</td>
</tr>
<tr>
<td>xsd:base64Binary</td>
<td>byte[]</td>
</tr>
<tr>
<td>xsd:hexBinary</td>
<td>byte[]</td>
</tr>
<tr>
<td>xsd:unsignedInt</td>
<td>long</td>
</tr>
<tr>
<td>xsd:unsignedShort</td>
<td>int</td>
</tr>
<tr>
<td>xsd:unsignedByte</td>
<td>short</td>
</tr>
<tr>
<td>xsd:time</td>
<td>javax.xml.datatype.XMLGregorianCalendar</td>
</tr>
<tr>
<td>xsd:date</td>
<td>javax.xml.datatype.XMLGregorianCalendar</td>
</tr>
<tr>
<td>xsd:g</td>
<td>javax.xml.datatype.XMLGregorianCalendar</td>
</tr>
<tr>
<td>xsd:anySimpleType</td>
<td>java.lang.Object</td>
</tr>
<tr>
<td>xsd:anySimpleType</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:duration</td>
<td>javax.xml.datatype.Duration</td>
</tr>
<tr>
<td>XML Schema Type</td>
<td>Java Data Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>xsd:NOTATION</td>
<td>javax.xml.namespace.QName</td>
</tr>
</tbody>
</table>
Tutorial: Using data types with Microsoft .NET

The following tutorial demonstrates how to use the SQL Anywhere web service datatype support from within Microsoft .NET using Visual C#.

To create SOAP and DISH services

1. At a command prompt, run the following command to start a personal web server. Replace samples-dir with the actual location of the sample database. The -xs http (port=80) option tells the database server to accept HTTP requests on port 80. If you already have a web server running on port 80, use another port number such as 8080 for this tutorial.

```
 dbeng11 -xs http(port=80) samples-dir\demo.db
```

2. Start Interactive SQL. Connect to the SQL Anywhere sample database as the DBA. Execute the following statements:

a. Define a SOAP service that lists the Employees table.

```
CREATE SERVICE "SASoapTest/EmployeeList"
TYPE 'SOAP'
AUTHORIZATION OFF
SECURE OFF
USER DBA
DATATYPE OUT
AS SELECT * FROM Employees;
```

In this example, DATATYPE OUT is specified to generate datatype 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' />
```

b. Create a DISH service to act as a proxy for the SOAP service and to generate the WSDL document.
CREATE SERVICE "SASoapTest_DNET"
TYPE 'DISH'
GROUP "SASoapTest"
FORMAT 'DNET'
AUTHORIZATION OFF
SECURE OFF
USER DBA;

The SOAP and DISH service must be of format DNET. In this example, the FORMAT clause was omitted when the SOAP service was created. As a result, the SOAP service inherits the DNET format from the DISH service.

3. Start Microsoft Visual Studio. Note that this example uses functions from the .NET Framework 2.0.
   a. Create a new Windows Application project using Visual C#.

       An empty form appears.

   b. From the Project menu, choose Add Web Reference.

   c. In the URL field of the Add Web Reference page, enter the following URL:

       http://localhost:80/demo/SASoapTest_DNET.

   d. Click Go.

       You are presented with a list of the methods available for SASoapTest_DNET. You should see the EmployeeList method.

   e. Click Add Reference to finish.

       The Solution Explorer window shows the new web reference.

   f. From the Visual Studio Toolbox, add a listbox and a button to the form as shown in the following diagram.

   g. Rename the button text to Employee List.
Double-click **Employee List** and add the following code for the button click event.

```csharp
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("\n");
```
dr.Close();

This example is designed to illustrate the fine control the application developer can have over the
datatype information that is available.

i. Build and run the program.

The XML response from the web server includes a formatted result set. The first row of the formatted result
set is shown below.

```xml
<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>45700.00</Salary>
  <StartDate>1984-08-28T05:00</StartDate>
  <BirthDate>1958-06-05T05:00</BirthDate>
  <BenefitHealthInsurance>1</BenefitHealthInsurance>
  <BenefitLifeInsurance>1</BenefitLifeInsurance>
  <BenefitDayCare>0</BenefitDayCare>
  <Sex>F</Sex>
</row>
```

There are a few things to note about the XML result set response.

- All column data is converted to a string representation of the data.
- Columns that contain date and/or time information include the offset from UTC of the web server. In
  this example, the offset is -05:00 which is 5 hours to the west of UTC (in this case, North American
  Eastern Standard Time).
- Columns containing date only are formatted as follows: `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 time only are formatted as follows: `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 date and time are formatted as follows: `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 listbox will display the EmployeeList result set as (type)column name=value pairs. The result from
processing the first row of the result set is shown below.

```plaintext
(Int32)EmployeeID=102
(Int32)ManagerID=501
(String)Surname=Whitney
(String)GivenName=Fran
(Int32)DepartmentID=100
```
There are a couple of things to note about the results.

- Columns that contain nulls are returned as type DBNull.
- The Salary amount has been converted to the client's currency format.
- Columns that contain date but no time values assume a time of 00:00:00 or midnight.
Tutorial: Using data types with JAX-WS

The following tutorial demonstrates how to access web services using the Java API for XML Web Services (JAX-WS).

If you have done the earlier JAX-WS tutorial, then some of these steps will already have been performed.

Before you begin, you will need the JAX-WS tools available from Sun. If you do not have this package installed on your system, you must download the JAX-WS tools and install them. To download the JAX-WS tools, visit http://java.sun.com/webservices/. Click the link for JAX-WS. This should take you to the Java API for XML Web Services page. Click the Download Now link. After you have downloaded the software package, install it on your system.

This example was developed with JAX-WS 2.1.3 for Windows.

SQL Anywhere SOAP web services that are accessed from JAX-WS should be declared to be of format CONCRETE.

To create SOAP and DISH services

1. At a command prompt, run the following command to start a personal web server. Replace samples-dir with the actual location of the sample database. The -xs http(port=80) option tells the database server to accept HTTP requests on port 80. If you already have a web server running on port 80, use another port number such as 8080 for this tutorial.

   dbeng11 -xs http(port=80) samples-dir\demo.db

2. Start Interactive SQL and connect to the SQL Anywhere sample database as the DBA. Execute the following statements:

   a. Define a stored procedure that lists some columns of the Employees table.

      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;

   b. Define a SOAP service that calls this stored procedure.

      CREATE SERVICE "WS/EmployeeList2" TYPE 'SOAP' FORMAT 'CONCRETE' EXPLICIT ON DATATYPE OUT AUTHORIZATION OFF SECURE OFF USER DBA AS CALL ListEmployees();
The EXPLICIT clause can only be used with a SOAP or DISH service of type CONCRETE. In this example, EXPLICIT ON indicates that the corresponding DISH service should generate XML Schema that describes the EmployeeList2Dataset object. This option only affects the WSDL document that is generated. In an earlier JAX-WS tutorial, we looked at an example that used EXPLICIT OFF. See “Tutorial: Accessing web services from JAX-WS” on page 871.

DATATYPE OUT indicates that explicit data type information is generated in the XML result set response. If DATATYPE OFF had been specified, then all data would be typed as string. This option does not affect the WSDL document that is generated.

Because authorization has been turned off, anyone can use this service without supplying a user name and password. The commands run under user DBA. This arrangement is simple, but not secure.

c. Create a DISH service to act as a proxy for the SOAP service and to generate the WSDL document.

```
CREATE SERVICE "WSDish"
TYPE 'DISH'
FORMAT 'CONCRETE'
GROUP "WS"
AUTHORIZATION OFF
SECURE OFF
USER DBA;
```

The SOAP and DISH service must both be format CONCRETE. Since the EmployeeList2 service is in the WS group, a GROUP clause is included.

3. Take a look at the WSDL that the DISH service automatically creates. To do so, open a web browser and browse to the following URL: http://localhost:80/demo/WSDish. The DISH service automatically generates a WSDL document that appears in the browser window.

In particular, observe the EmployeeList2Dataset object that is exposed because the format of this service is CONCRETE and EXPLICIT is ON. In a later step, the wsimport application uses this information to generate a SOAP 1.1 client interface for this service.

```xml
<s:complexType name="EmployeeList2Dataset">
  <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>
```

To generate a JAX-WS interface for these web services

1. In this example, the Java API for XML Web Services (JAX-WS) and the Sun Java 1.6.0 JDK are installed on drive C:. Set your PATH environment variable so that it includes the JAX-WS binaries and the JDK binaries. The binaries are located in the following directories:

   c:\Sun\jaxws-ri\bin
   c:\Sun\SDK\jdk\bin

2. At a command prompt, set your CLASSPATH environment variable.

   SET classpath=.;C:\Sun\jaxws-ri\lib\jaxb-api.jar
     ;C:\Sun\jaxws-ri\lib\jaxws-rt.jar

3. The next step is to generate the interface needed to call the web service.

   At the same command prompt, create a new project directory and make this directory your current directory. Execute the following command in this directory.

   wsimport -keep -Xendorsed "http://localhost:80/demo/WSDish"

   The wsimport tool retrieves the WSDL document from the given URL, generates the Java files that define the interface for it, and then compiles the Java files.

   The keep option tells wsimport not to delete the .java files. Without this option, these files are deleted after the corresponding .class files have been generated. Saving these files makes it easier to examine the makeup of the interface.

   The Xendorsed option allows you to use the JAX-WS 2.1 API with JDK6.

   Once this command completes, you should have a new subdirectory structure named localhost\demo\ws in your current directory that contains the following Java files, along with the compiled .class versions of each Java file.

   EmployeeList2.java
   EmployeeList2Dataset.java
   EmployeeList2Response.java
   FaultMessage.java
   ObjectFactory.java
   package-info.java
   WSDish.java
   WSDishSoapPort.java

To use the generated JAX-WS interface

1. Save the following Java source code into SASoapDemo2.java. Make sure that this file is located in the same directory containing the localhost subdirectory that was generated by the wsimport tool.

   // SASoapDemo2.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.demo.ws.*;

public class SASoapDemo2
{
    public static void main( String[] args )
    {
        try {
            WSDish service = new WSDish();

            Holder<EmployeeList2Dataset> response =
                new Holder<EmployeeList2Dataset>();
            Holder<Integer> sqlcode = new Holder<Integer>();

            WSDishSoapPort port = service.getWSDishSoap();

            // This is the SOAP service call to EmployeeList2
            port.employeeList2( response, sqlcode );

            EmployeeList2Dataset result = response.value;
            EmployeeList2Dataset.Rowset rowset = result.getRowset();

            List<EmployeeList2Dataset.Rowset.Row> rows = rowset.getRow();

            String fieldType;
            String fieldName;
            String fieldValue;
            Integer fieldInt;
            XMLGregorianCalendar fieldDate;

            for ( int i = 0; i < rows.size(); i++ ) {
                EmployeeList2Dataset.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().getSimpleType();
                    fieldName = "TerminationDate";
                    fieldValue = row.getTerminationDate().getValue();
                }
            }
        } catch ( Exception e ) {
            System.out.println( e.getMessage() );
        }
    }
}
If you chose to start the web server using a different port number such as 8080, then you will have to alter the import localhost source line to something like the following:

```
import localhost._8080.demo.ws.*;
```

2. Compile `SASoapDemo2.java`.

```
javac SASoapDemo2.java
```

3. Run the compiled class file.

```
java SASoapDemo2
```

When the application sends its request to the web server, it receives an XML result set response that consists of an EmployeeList2Result with a rowset containing several row entries. Also included in the response is the sqlcode result from executing the query. An example of the response is shown next.

```
<tns:EmployeeList2Response>
  <tns:EmployeeList2Result xsi:type='tns:EmployeeList2Dataset'>
    <tns:rowset>
      <tns:row>...</tns:row>
      ...
      ...
      <tns:row>...</tns:row>
    </tns:rowset>
  </tns:EmployeeList2Result>
  <tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeList2Response>
```

Each row in the rowset is sent in a format similar to the following.

```
<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:date">1994-07-12-04:00</tns:StartDate>
  <tns:TerminationDate xsi:type="xsd:date">2008-04-18-04:00</tns:TerminationDate>
</tns:row>
```

Note that the column name and data type are included.

**Using a proxy**

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.
The EmployeeList2 result set is displayed as (type)column name=value pairs by the SASoapDemo2 application. Several lines of output similar to the following should be generated.

```plaintext
(Integer) EmployeeID=102
(String) Surname=Whitney
(String) GivenName=Fran
(XMLGregorianCalendar) StartDate=1984-08-28-04:00
(unknown) TerminationDate=null

(Integer) EmployeeID=105
(String) Surname=Cobb
(String) GivenName=Matthew
(XMLGregorianCalendar) StartDate=1985-01-01-05:00
(unknown) TerminationDate=null

(Integer) EmployeeID=1740
(String) Surname=Nielsen
(String) GivenName=Robert
(XMLGregorianCalendar) StartDate=1994-06-24-04:00
(unknown) TerminationDate=null

(Integer) EmployeeID=1751
(String) Surname=Ahmed
(String) GivenName=Alex
(XMLGregorianCalendar) StartDate=1994-07-12-04:00
(XMLGregorianCalendar) TerminationDate=2008-04-18-04:00
```

Note that 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.

Also note that date values include an offset from UTC time. In the above sample data, the server is located in the North American Eastern time zone. This is 5 hours earlier than UTC time (-05:00) for dates when daylight savings is not in effect and 4 hours earlier than UTC time (-04:00) for dates when daylight saving is in effect.

For further understanding of the Java methods used in the SASoapDemo2 application, it is worthwhile investigating the javax.xml.bind.JAXBElement class documentation on the java.sun.com web site (http://java.sun.com/javaee/5/docs/api/).
Using the iAnywhere WSDL compiler

Given a WSDL source that describes a web service, 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.

The Java or C# classes generated by the WSDL compiler are intended for use with QAnywhere. These classes expose web service operations 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:

- .NET: “QAnywhere .NET API for web services (.NET 2.0)” [QAnywhere]
- Java: “QAnywhere Java API for web services” [QAnywhere]

The WSDL compiler supports WSDL 1.1 and SOAP 1.1 over HTTP and HTTPS.

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.

This is the name of the WSDL file that describes a web service.

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.
Creating web service client functions and procedures

A SQL Anywhere database can provide and consume web services. These can be standard web services available over the Internet, or these can be provided by SQL Anywhere databases, as long as the web service is not in the same database as the client procedure or function.

SQL Anywhere can act as both HTTP and SOAP web service clients. This functionality is provided through stored functions and stored procedures.

Client functions and procedures are created and manipulated using the following SQL statements:

- “CREATE FUNCTION statement (web services)” [SQL Anywhere Server - SQL Reference]
- “CREATE PROCEDURE statement (web services)” [SQL Anywhere Server - SQL Reference]
- “ALTER FUNCTION statement” [SQL Anywhere Server - SQL Reference]
- “ALTER PROCEDURE statement” [SQL Anywhere Server - SQL Reference]
- “DROP FUNCTION statement” [SQL Anywhere Server - SQL Reference]
- “DROP PROCEDURE statement” [SQL Anywhere Server - SQL Reference]

For example, the syntax of the CREATE FUNCTION and CREATE PROCEDURE statements, as used to create web service client functions, are as follows:

```
CREATE FUNCTION [ owner.]procedure-name ( [ parameter, ... ] )
RETURNS data-type
URL url-string
[ proc-attributes ]
```

```
CREATE PROCEDURE [ owner.]procedure-name ( [ parameter, ... ] )
URL url-string
[ proc-attributes ]
```

Key to this syntax is the URL clause, which is used to provide the URL of the web service that you want the procedure to access. The basic syntax of the URL clause is as follows:

```
url-string :
'[{ HTTP | HTTPS | HTTPS_FIPS }]://[ user:password@ ]hostname[ :port ][/path ]'
```

The optional user and password information permits you to access web services that require authentication. The hostname can be the name or the IP address of the computer providing the web service.

The port number is required only if the server is listening on a port number other than the default. The default port numbers are 80 for HTTP services and 443 for HTTPS services.

The path identifies the resource or web service on the server.

A request can be sent to any web service, whether it is provided by another SQL Anywhere database or available over the Internet. If the web service is provided by the same database server, it cannot reside in the same database as the client function. Attempting to access a web service in the same database results in the error 403 Forbidden.

Because it is used for parameter substitution, exclamation marks that appear in strings anywhere in the procedure definition must be escaped. See “Escaping the ! character” on page 910.
Common clauses

The additional clauses that enable you to supply more details about the procedure call are as follows:

```
proc-attributes :
  [ TYPE { 'HTTP': [ GET | POST | PUT | DELETE | HEAD ] } | 'SOAP': [ RPC | DOC ] ]
  [ NAMESPACE namespace-string ]
  [ CERTIFICATE certificate-string ]
  [ CLIENTPORT clientport-string ]
  [ PROXY proxy-string ]
  [ SET protocol-option-string ]
```

The TYPE clause is important because it tells SQL Anywhere how to format the request to the web service provider. Standard SOAP types RPC and DOC are available. The standard HTTP methods of GET and POST are available, specified as HTTP:GET and HTTP:POST, respectively. Specifying HTTP implies HTTP:POST.

If type SOAP is chosen, SQL Anywhere automatically formats the request as an XML document in the standard format necessary for SOAP requests. Since a SOAP request is always an XML document, an HTTP POST request is always implicitly used to send the SOAP request document to the server when type SOAP is chosen. Specifying SOAP implies SOAP:RPC.

Names for web service client functions and procedures

The procedure name is used as the SOAP operation name when building the outgoing SOAP request. In addition, the names of any parameters also appear in tag names in the SOAP request envelope. Since the SOAP server expects to see these names, specifying them correctly is an important part of defining a SOAP stored procedure. This fact places restrictions on the name of SOAP procedures and functions, beyond the general rules that apply to procedure and function names in SQL Anywhere.

The following statement creates a SOAP stored procedure named MyOperation:

```
CREATE PROCEDURE MyOperation ( a INTEGER, b CHAR(128) )
URL 'HTTP://localhost'
TYPE 'SOAP:DOC';
```

When this procedure is called, such as by the following statement, a SOAP request is generated:

```
CALL MyOperation( 123, 'abc' );
```

The procedure name appears in the `<m:MyOperation>` tag within the request body. The two parameters to the procedure, a and b, become `<m:a>` and `<m:b>`, respectively.

```
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost">
  <SOAP-ENV:Body>
    <m:MyOperation>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:MyOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
**Namespace URIs**

All SOAP requests require a method namespace URI. The SOAP processor on the server side uses this URI to understand the names of the various entities in the message body of the request.

When creating a SOAP function or procedure, of either SOAP:DOC or SOAP:RPC, you may be required to specify a namespace URI before the call succeeds. You can obtain the required namespace value from the WSDL description document, or from other available documentation for the service. The NAMESPACE clause applies only to SOAP functions and procedures. The default namespace value is the procedure's URI, up to, but not including, the optional path component and excluding any user and password values.

**HTTPS requests**

To issue a secure HTTPS request, the client must have access to the server's certificate, or the certificate used to sign the server's certificate. This certificate tells SQL Anywhere how to encrypt the request. Certificate values are also required when a request directed to an insecure server might be redirected to a secure one.

There are two ways to provide the certificate information. You can either place the certificate in a file and provide the file name, or you can provide the entire certificate as a string value. You cannot do both.

The certificate attributes are supplied as a string value constructed as key=value pairs separated by semicolons:

```
certificate-string : 
{ file=filename | certificate=string } ; company=company ; unit=company-unit ; name=common-name
```

The following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td></td>
<td>The file name of the certificate.</td>
</tr>
<tr>
<td>certificate</td>
<td>cert</td>
<td>The certificate itself, Base64 encoded.</td>
</tr>
<tr>
<td>company</td>
<td>co</td>
<td>The company specified in the certificate.</td>
</tr>
<tr>
<td>unit</td>
<td></td>
<td>The company unit specified in the certificate.</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>The common name specified in the certificate.</td>
</tr>
</tbody>
</table>

For example, the following statement creates a procedure that makes a secure request to a web service located on the same computer as the client:

```
CREATE PROCEDURE test()
URL 'HTTPS://localhost/myservice'
CERTIFICATE 'file=C:\srv_cert.id;co=iAnywhere;
unit=SA;name=JohnSmith';
```

Since no TYPE clause was provided, the request is assumed to be of type SOAP:RPC. The server's public certificate is located in the file `C:\srv_cert.id`. 

902 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
Client ports

When accessing web services through a firewall, it is sometimes necessary to tell SQL Anywhere which ports to use when opening a connection to the server. Ordinarily, port numbers are obtained dynamically, and you should rely on the default behavior unless your firewall restricts access to a particular range of ports.

The ClientPort option designates the port number on which the client application communicates using TCP/IP. You can specify a single port number, or a combination of individual port numbers, and ranges of port numbers, as demonstrated in the following example:

```sql
CREATE PROCEDURE test ()
URL 'HTTPS://localhost/myservice'
CLIENTPORT '5040,5050-5060,5070';
```

It is best to specify a list or a range of port numbers. If you specify a single port number, then only one connection is maintained at a time. In fact, even after closing the one connection, there is a timeout period of several minutes during which no new connection can be made to the same remote server and port. When you specify a list and/or range of port numbers, the application keeps trying port numbers until it finds one to which it can successfully bind.

This feature is similar to the ClientPort network protocol option. See “ClientPort protocol option [CPORT]” [SQL Anywhere Server - Database Administration].

Using proxies

Some web service requests may need to be made through a proxy server. When this is the case, the URL of the proxy server must be supplied using the PROXY clause.

The format of the value is the same as for the URL clause, although any user, password, or path values are ignored:

```
proxy-string :
 '{ [ HTTP | HTTPS ]:[ ]user:password@ ]hostname[ :port ][ /path ]'
```

When a proxy server is specified, SQL Anywhere formats the request and sends it to the proxy server using the supplied proxy URL. The proxy server forwards the request to the final destination, obtains the response, and forwards the response back to SQL Anywhere.

Logging web service client procedures

Information from web service clients, including HTTP requests and transport data, can be logged to the web service client log file. You enable logging to this file by starting the database server with the -zoc server option or by setting the WebClientLogging server property using the sa_server_option system procedure. See “-zoc server option” [SQL Anywhere Server - Database Administration], and “sa_server_option system procedure” [SQL Anywhere Server - SQL Reference].

Modifying HTTP headers

When creating a web service procedure using the CREATE PROCEDURE statement, if you specify an HTTP HEADER name with no colon (:) and no value, the HTTP client application suppresses the header. If you include a colon but do not provide a value, the HTTP client application includes the header name, but no value. For example:
CREATE PROCEDURE suds(...)
   TYPE 'SOAP:RPC'
   URL '...'
   HEADER 'SOAPAction
      Date
      From:';

In this example, the Action and Date HTTP headers, which are automatically generated by SQL Anywhere, are suppressed, and the From header is included but with no value.

Modifying automatically generated headers, can have unexpected results. The following HTTP headers are typically automatically generated and should not be modified without very careful consideration.

<table>
<thead>
<tr>
<th>HTTP header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-Charset</td>
<td>Always automatically generated. If changed or deleted, may result in unexpected data conversion errors.</td>
</tr>
<tr>
<td>ASA-Id</td>
<td>Always automatically generated. Ensures client does not connect to itself (that is to the same server) which may result in deadlock.</td>
</tr>
<tr>
<td>Authorization</td>
<td>Automatically generated when URL contains credentials. If changed or deleted, may result in failure of the request. Only BASIC authorization is supported. User and password information should only be included when connecting via HTTPS.</td>
</tr>
<tr>
<td>Connection</td>
<td>Connection: close, is always automatically generated. Client does not support persistent connections. Should not be changed or connection may hang.</td>
</tr>
<tr>
<td>Host</td>
<td>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.</td>
</tr>
<tr>
<td>Transfer-Encoding</td>
<td>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.</td>
</tr>
<tr>
<td>Content-Length</td>
<td>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.</td>
</tr>
</tbody>
</table>
Working with return values and result sets

Web service client calls can be made with either stored functions, or stored procedures. If made from a function, the return type of the function must be of a character data type, such as CHAR, VARCHAR, or LONG VARCHAR. The value returned is the body of the HTTP response. 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 this additional information is wanted.

SOAP procedures

The response from a SOAP function is an XML document that contains the SOAP response.

Since SOAP responses are structured XML documents, 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 of the subtree below each of these tags is used as the row value for that column.

For example, given the SOAP response shown below, SQL Anywhere would construct the shown data set:

```xml
<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>
```

<table>
<thead>
<tr>
<th>Eliza</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi, I'm Eliza. Nice to meet you.</td>
</tr>
</tbody>
</table>

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 has only 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 also be accessed using the built-in Open XML processing capabilities.

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:

```sql
CREATE SERVICE get_webservices
    TYPE 'SOAP'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM SYSWEBSERVICE;
```
The following stored 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, as 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 demonstrates 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 service exists, this function returns the following result set:

<table>
<thead>
<tr>
<th>Name</th>
<th>Secure?</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_webservices</td>
<td>N</td>
</tr>
</tbody>
</table>

For more information about the XML processing facilities available in SQL Anywhere, see “Using XML in the database” [SQL Anywhere Server - SQL Usage].

### 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 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>HTTP /1.0 200 OK</td>
</tr>
<tr>
<td>Body</td>
<td>&lt;!DOCTYPE HTML ... &gt; &lt;HTML&gt; ... &lt;/HTML&gt;</td>
</tr>
<tr>
<td>Content-Type</td>
<td>text/html</td>
</tr>
<tr>
<td>Server</td>
<td>GWS/2.1</td>
</tr>
<tr>
<td>Content-Length</td>
<td>2234</td>
</tr>
<tr>
<td>Date</td>
<td>Mon, 18 Oct 2004, 16:00:00 GMT</td>
</tr>
</tbody>
</table>
Selecting from result sets

The SELECT statement is used to retrieve values from result sets. Once retrieved, these values can be stored in tables or used to set variables.

```
CREATE PROCEDURE test( INOUT parm CHAR(128) )
URL 'HTTP://localhost/test'
TYPE 'HTTP';
```

Because it is of type HTTP, this procedure returns the two-column result set described in the previous section. In the first column is an attribute name; in the second column the attribute value. The keywords are as in the HTTP response header fields. A Body attribute contains the body of the message, which is typically an HTML document.

One approach is to insert the result sets into a table, such as the following one:

```
CREATE TABLE StoredResults(
   Attribute LONG VARCHAR,
   Value     LONG VARCHAR
);
```

Result sets can be inserted into this 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.
Using parameters

Stored functions and stored procedures that act as web service clients can be declared with parameters, just like other types of function or procedures. These parameter values, unless used during parameter substitution, are passed as part of the HTTP or SOAP request.

In addition, 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 no placeholders for a particular variable 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 web service request.

Passed parameters

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.

HTTP requests

Parameters to requests of type HTTP:GET are encoded in the URL. For example, the following procedure declares two parameters:

```sql
CREATE PROCEDURE test ( a INTEGER, b CHAR(128) )
URL 'HTTP://localhost/myservice'
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/myservice?a=123&b=xyz
```

If the type is HTTP:POST, the parameters and their values instead become part of the body of the request. In the case of the two parameter and values, the following text appears, after the headers, in the body of the HTTP request:

```
a=123&b=xyz
```

SOAP requests

Parameters passed to SOAP requests are bundled as part of the request body, as required by the SOAP specification:

```xml
<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">
 <SOAP-ENV:Body>
  <m:test>
   <m:a>123</m:a>
   <m:b>abc</m:b>
  </m:test>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Parameter substitution

Declared parameters to the stored procedure or stored function are automatically substituted for placeholders within the stored function or stored procedure definition each time the procedure or function is executed. Any substrings that contain an exclamation mark (!) followed by the name of one of the declared parameters are replaced by that parameter’s value.

For example, the following procedure definition permits the entire URL to be passed as a parameter. Different values can be used each time this procedure is called.

```sql
CREATE PROCEDURE test ( url CHAR(128) )
URL '!url'
TYPE 'HTTP:POST';
```

For example, you could then use the procedure as follows:

```sql
CALL test ( 'HTTP://localhost/myservice' );
```

Hiding user and password values

One useful application of parameter substitution is to avoid making sensitive values, such as user names and passwords, part of a web service client function or procedure definition. When such values are specified as literals in the procedure or function definition, they are stored in the system tables, making them readily accessible to all users of the database. Passing these values as parameters circumvents this problem.

For example, the following procedure definition includes the user name and password as plain text as part of the procedure definition:

```sql
CREATE PROCEDURE test
URL 'HTTP://dba:sql@localhost/myservice';
```

To avoid this problem, you can declare user and password as parameters. Doing so permits the user and password values to be supplied only when the procedure is invoked. For example:

```sql
CREATE PROCEDURE test ( uid CHAR(128), pwd CHAR(128) )
URL 'HTTP://!uid:!pwd@localhost/myservice';
```

This procedure is called as follows:

```sql
CALL test ( 'dba', 'sql' );
```

As another example, you can use parameter substitution to pass encryption certificates from a file and pass them to a stored procedure or stored function:

```sql
CREATE PROCEDURE secure( cert LONG VARCHAR )
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Server';
```

When you call this procedure, you supply the certificate as a string. In the following example call, the certificate is read from a file. This is done for illustration only, as the certificate can be read directly from a file using the file= keyword for the CERTIFICATE clause.

```sql
CALL secure( xp_read_file('install-dir\bin32\rsaserver.id') );
```
Escaping the ! character

Because the exclamation mark (!) is used to identify placeholders for parameter substitution in the context of web service client stored functions and stored procedures, it must be escaped whenever you want to include this character as part of any of the procedure attribute strings string. To do so, prefix the exclamation mark with a second exclamation mark. So, all occurrences of !! in strings within a web service client or web service function definition are replaced by !.

Parameter names used as placeholders must contain only alphanumeric characters. In addition, placeholders must be followed by a non-alphanumeric character to avoid ambiguity. 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:

```sql
CREATE PROCEDURE orderitem ( size CHAR(18) )
URL 'HTTP://salesserver/order?size=!sizeXL'
TYPE 'SOAP:RPC';
```

Instead, !sizeXL is always deleted because it is a valid placeholder for which there is no matching parameter.

Parameter data type conversion

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.
Working with 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.

For simple return data types, a string representation within a result set may suffice. In such a case, the use of a stored procedure may be warranted.

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 parameters of type XML, 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.

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.

<table>
<thead>
<tr>
<th>SQL Anywhere XML Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
</tr>
<tr>
<td>xsi</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
</tr>
<tr>
<td>SOAP-ENC</td>
<td><a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a></td>
</tr>
<tr>
<td>m</td>
<td>namespace as defined in the NAMESPACE clause</td>
</tr>
</tbody>
</table>

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 examples are designed to issue requests to the Microsoft SOAP ToolKit 3.0 Round 2 Interoperability test server (http://mssoapinterop.org/stkV3). 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.

All three 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:

```
CALL sa_make_object('function', 'echoFloatArray');
ALTER FUNCTION echoFloatArray( inputFloatArray XML )
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CALL sa_make_object('function', 'echoStruct');
ALTER FUNCTION echoStruct( inputStruct XML )
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CALL sa_make_object('function', 'echoStructArray');
ALTER 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[]' 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.
The response from the Interoperability server is shown below.

```xml
<?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()' );
```

<table>
<thead>
<tr>
<th>varFloat</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.9990005493</td>
</tr>
<tr>
<td>199.9989929199</td>
</tr>
</tbody>
</table>

2. The parameters in the following example represent a structure.

```sql
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.

```xml
<inputStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
</inputStruct>
```

The response from the Interoperability server is shown below.
If the response was stored in a variable, then it can be parsed using OPENXML.

```sql
SELECT * FROM openxml( resp,'//*:Body/*:SOAPStruct' )
WITH ( varString LONG VARCHAR 'varString',
      varInt INT 'varInt',
      varFloat FLOAT 'varFloat' );
```

<table>
<thead>
<tr>
<th>varString</th>
<th>varInt</th>
<th>varFloat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ninety-Nine</td>
<td>99</td>
<td>99.9990005493</td>
</tr>
</tbody>
</table>

3. The parameters in the following example represent an array of structures.

```sql
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.

```xml
<inputStructArray xmlns:q2="http://soapinterop.org/xsd" SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
  </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>99.9990005493</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.

```sql
SELECT * FROM openxml( resp,'//*:Body/*:SOAPStruct' )
WITH ( varString LONG VARCHAR 'varString', varInt INT 'varInt', varFloat FLOAT 'varFloat' );
```

<table>
<thead>
<tr>
<th>varString</th>
<th>varInt</th>
<th>varFloat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ninety-Nine</td>
<td>99</td>
<td>99.9990005493164</td>
</tr>
<tr>
<td>varString</td>
<td>varInt</td>
<td>varFloat</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Hundred and Ninety-Nine</td>
<td>199</td>
<td>199.9989929199</td>
</tr>
</tbody>
</table>
Working with variables

Variables in HTTP requests come from one of two sources. First, the URL can include a query string, which includes various name=value pairs. HTTP GET requests are formatted in this manner. Here is an example of a URL that contains a query string.

http://localhost/gallery?picture=sunset.jpg

The second way is through the URL path. Setting URL PATH to either ON or ELEMENTS causes the portion of the path following the service name to be interpreted as variable values. This option allows URLs to appear to be requesting a file in a particular directory, as would be the case on a traditional file-based web site, rather than something stored inside a database. The following is an example.

http://localhost/gallery/sunset.jpg

This URL appears to request the file sunset.jpg from a directory named gallery. In fact, the gallery service receives this string as a parameter (using it, perhaps, to retrieve a picture from a database table).

The parameter passed in HTTP requests depends on the setting of URL PATH.

- **OFF**  No path parameters are permitted after the service name.
- **ON**   All path elements after the service name are assigned to the variable URL.
- **ELEMENTS**  The remainder of the URL path is split at the slash characters into a list of up to 10 elements. These values are assigned the variables URL1, URL2, URL3, ..., URL10. If there are fewer than 10 values, the remaining variables are set to NULL. Specifying more than ten variables causes an error.

Apart from the location in which they are defined, there is no difference between variables. You access and use all HTTP variables the same way. For example, the values of variables such as url1 are accessed in the same way as parameters that appear as part of a query, such as ?picture=sunset.jpg.

Accessing variables

There are two main ways to access variables. The first is to mention variables in the statement of the service declaration. For example, the following statement passes the value of multiple variables to the ShowTable stored procedure:

```
CREATE SERVICE ShowTable
  TYPE 'RAW'
  AUTHORIZATION ON
AS CALL ShowTable( :user_name, :table_name, :limit, :start );
```

The other way is to use the built-in functions NEXT_HTTP_VARIABLE and HTTP_VARIABLE within the stored procedure that handles the request. If you do not know which variables are defined, you can use the NEXT_HTTP_VARIABLE to find out. The HTTP_VARIABLE function returns the variable values.

The NEXT_HTTP_VARIABLE function allows you to iterate through the names of the defined variables. The first time you call it, you pass in the NULL value. This returns the name of one variable. Calling it subsequent times, each time passing in the name of the previous variable, returns the next variable name. When the name of the last variable is passed to this function, it returns NULL.
Iterating through the variable names in this manner guarantees that each variable name is returned exactly once. However, the order that the values are returned may not be the same as the order that they appear in the request. In addition, if you iterate through the names a second time, they can be returned in a different order.

To get the value of each variable, use the HTTP_VARIABLE function. The first parameter is the name of the variable. Additional parameters are optional. In the case that multiple values were supplied for a variable, the function returns the first value if supplied with only one parameter. Supplying an integer as the second parameter allows you to retrieve additional values.

The third parameter allows you to retrieve variable header-field values from multi-part requests. Supply the name of a header field to retrieve its value. For example, the following SQL statements retrieve three variable values, then retrieve the header-field values of the image variable.

```sql
SET v_id = HTTP_VARIABLE( 'ID' );
SET v_title = HTTP_VARIABLE( 'Title' );
SET v_descr = HTTP_VARIABLE( 'descr' );

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' );
```

Here is an example that uses the HTTP_VARIABLE function to retrieve the values associated with the variables. It is a modified version of the ShowSalesOrderDetail service described in an earlier section.

```sql
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 service that invokes the stored procedure follows:

```sql
CREATE SERVICE ShowDetail
TYPE 'HTML'
URL PATH OFF
AUTHORIZATION OFF
USER DBA
AS CALL ShowDetail();
```

To test the service, open a web browser and supply the following URL:

http://localhost:80/demo/ShowDetail?product_id=300&customer_id=101

For more information about parameter passing, see “Understanding how URLs are interpreted” on page 861 and “Web services functions” [SQL Anywhere Server - SQL Reference].
Working with HTTP headers

Server side

When creating HTTP web service client procedures, use this clause to add, modify, or delete HTTP request header entries. The specification of headers closely resembles the format specified in RFC2616 Hypertext Transfer Protocol — HTTP/1.1, and RFC822 Standard for ARPA Internet Text Messages, including the fact that only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive. Following are a few key points regarding HTTP header specification:

- Header/value pairs can be delimited by \n or \x0d\n, specifying Line Feed (<LF>), or Carriage Return and Line Feed(<CR><LF>), respectively.
- A header is delimited from its value using a colon (:), and therefore cannot contain a colon.
- A header followed by \n, or an end of line, specifies a header with no value, as does a header with no colon or value after it. For example, HEADER 'Date', specifies that the Date header should not be included. Suppressing headers, or their values, can cause unexpected results. See "Modifying HTTP headers" on page 903.
- Folding of long header values is supported, provided one or more white spaces immediately follow the \n. For example, the following HEADER specification, and resulting HTTP output, are semantically equivalent:

  HEADER 'heading1: This long value
  heading2:shortvalue'

  HEADER 'heading1:This long value is a really long value for heading1
  heading2:shortvalue'<

- Multiple contiguous white spaces, including folding, results in a single white space.
- Parameter substitution is supported for this clause.

This example shows how to add static user-defined headers:

```sql
CREATE PROCEDURE http_client()
    URL 'http://localhost/getdata'
    TYPE 'http:get'
    HEADER 'UserHeader1:value1
UserHeader2:value2';
```

This example shows how to add parameter-substituted user-defined headers:

```sql
CREATE PROCEDURE http_client( headers LONG VARCHAR )
    URL 'http://localhost/getdata'
    TYPE 'http:get'
    HEADER '!headers';

CALL http_client( 'NewHeader1:value1
NewHeader2:value2' );
```

Client side

Headers in HTTP requests can be obtained using a combination of the NEXT_HTTP_HEADER and HTTP_HEADER functions. The NEXT_HTTP_HEADER function iterates through the HTTP headers included within a request and returns the next HTTP header name. Calling it with NULL causes it to return
the name of the first header. Subsequent headers are retrieved by passing the function the name of the previous
header. This function returns NULL when called with the name of the last header.

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order
that they appear in the HTTP request.

The HTTP_HEADER function returns the value of the named HTTP header field, or NULL if not called
from an HTTP service. It is used when processing an HTTP request via a web service. If a header for the
given field name does not exist, the return value is NULL.

Here is a table of some typical HTTP headers and values.

<table>
<thead>
<tr>
<th>Header Name</th>
<th>Header Value</th>
</tr>
</thead>
</table>
| 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                                                                          |
| UA-CPU      | x86                                                                           |
| 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                                                                    |
| @HttpMethod | GET                                                                           |
| @HttpURI    | /demo/ShowHTTPHeaders                                                        |
| @HttpVersion| HTTP/1.1                                                                      |

To get the value of each header, use the NEXT_HTTP_HEADER function to get the name of the header
and then use the HTTP_HEADER function to get its value. The following example illustrates how to do
this.

```sql
CREATE PROCEDURE HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    declare header_name long varchar;
    declare header_value 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
            break
            SET header_value = HTTP_HEADER( header_name );
            table_rows = table_rows + XML Wochen:
                <header>
                    <name>
                        %header_name%
                    </name>
                    <value>
                        %header_value%
                    </value>
                </header>
            END IF
        END LOOP
    RETURN HTML(table_rows);
END;
```
This example formats the header names and values into an HTML table. The following service can be defined to show how this sample procedure works.

```sql
CREATE SERVICE ShowHTTPHeaders
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS CALL HTTPHeaderExample();
```

To test the service, open a web browser and supply the following URL:

http://localhost:80/demo/ShowHTTPHeaders

To set the status code (or response code) of the request being processed, use the @HttpStatus special header. See “sa_set_http_header system procedure” [SQL Anywhere Server - SQL Reference].

For more information about header processing, see “Web services functions” [SQL Anywhere Server - SQL Reference].
Using SOAP services

To illustrate many of the features of web services, start with a simple Fahrenheit to Celsius temperature convertor as a sample service.

To set up a simple web service server

1. Create a database.
   
   ```
   dbinit ftc
   ```

2. Start a server using this database.
   
   ```
   dbsrv11 -xs http(port=8082) -n ftc ftc.db
   ```

3. Connect to the server using Interactive SQL.
   
   ```
   dbisql -c "UID=DBA;PWD=sql;ENG=ftc"
   ```

4. Using Interactive SQL, create a web service.
   
   ```
   CREATE SERVICE FtoCService 
   TYPE 'SOAP' 
   FORMAT 'XML' 
   AUTHORIZATION OFF 
   USER DBA 
   AS CALL FToCConvertor( :temperature );
   ```

5. Define the stored procedure that this service is to call to perform the calculation needed to convert from a temperature expressed in degrees Fahrenheit to a temperature expressed in degrees Celsius:

   ```
   CREATE PROCEDURE FToCConvertor( temperature FLOAT ) 
   BEGIN 
   SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) 
   AS answer; 
   END;
   ```

At this point, you now have a SQL Anywhere web service server running and ready to handle requests. The server is listening for SOAP requests on port 8082.

So how can you test this SOAP request server? The simplest way to do this is to use another SQL Anywhere database server to communicate the SOAP request and retrieve the response.

To send and receive SOAP requests

1. Create another database for use with a second server.
   
   ```
   dbinit ftc_client
   ```

2. Start the personal server using this database.
   
   ```
   dbeng11 ftc_client.db
   ```

3. Connect to the personal server using another instance of Interactive SQL.
   
   ```
   dbisql -c "UID=DBA;PWD=sql;ENG=ftc_client"
   ```

4. Using Interactive SQL, create a stored procedure.
CREATE PROCEDURE FtoC( temperature FLOAT )
URL 'http://localhost:8082/FtoCService'
TYPE 'SOAP:DOC';

The URL clause is used to reference the SOAP web service. The string 'http://localhost:8082/FtoCService' specifies the URI of the web service that is going to be used. This is a reference to the web server that is listening on port 8082.

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 data types. SOAP requests are always sent as XML documents. The mechanism for sending SOAP requests is 'HTTP:POST'.

5. You need a wrapper for the FtoC stored procedure, so create a second stored procedure.

```
CREATE PROCEDURE FahrenheitToCelsius( temperature FLOAT )
BEGIN
    DECLARE result LONG VARCHAR;
    DECLARE err INTEGER;
    DECLARE crsr CURSOR FOR
        CALL FtoC( temperature );

    OPEN crsr;
    FETCH crsr INTO result, err;
    CLOSE crsr;

    SELECT temperature, Celsius
    FROM OPENXML(result, '//tns:answer', 1, result)
    WITH ("Celsius" FLOAT 'text()');
END;
```

This stored procedure acts as a cover procedure for the call to the web service. The FtoC stored procedure returns a result set that this stored procedure processes. The result set is a single XML string that looks like the following.

```
<tns:rowset xmlns:tns="http://localhost/ftc/FtoCService">
    <tns:row>
        <tns:answer>100</tns:answer>
    </tns:row>
</tns:rowset>
```

The OPENXML function is used to parse the XML that is returned, extracting the value that is the temperature in degrees Celsius.

6. Call the stored procedure to send the request and obtain the response.

```
CALL FahrenheitToCelsius(212);
```

The Fahrenheit temperature and the Celsius equivalent appear.

<table>
<thead>
<tr>
<th>temperature</th>
<th>Celsius</th>
</tr>
</thead>
<tbody>
<tr>
<td>212</td>
<td>100</td>
</tr>
</tbody>
</table>

At this point, a simple web service running on a SQL Anywhere web server has been demonstrated. As you have seen, other SQL Anywhere servers can communicate with this web server. There has been little control...
over the content of the SOAP requests and responses that have traveled between these servers. In the next section, you will see how this simple web service can be extended by adding your own SOAP headers.

**Note**
The web service can be provided by the same database server, but must not reside in the same database as the client function. Attempting to access a web service in the same database results in the error 403 Forbidden.

For information about SOAP header processing, see “Working with SOAP headers” on page 925.
Working with SOAP headers

In this section, the simple web service that was introduced in “Using SOAP services” on page 922 is extended to handle SOAP headers.

If you have followed the steps outlined in the previous section, you can skip steps 1 through 4 and go directly to step 5.

To create a web service server

1. Create a database.
   
   `dbinit ftc`

2. Start a server using this database.
   
   `dbsrv11 -xs http(port=8082) -n ftc ftc.db`

3. Connect to the server using Interactive SQL.
   
   `dbisql -c "UID=DBA;PWD=sql;ENG=ftc`"

4. Using Interactive SQL, create a web service.
   
   ```
   CREATE SERVICE FtoCService
   TYPE 'SOAP'
   FORMAT 'XML'
   AUTHORIZATION OFF
   USER DBA
   AS CALL FToCConvertor( :temperature );
   ```

5. Define the stored procedure that this service is to call to perform the calculation needed to convert from a temperature expressed in degrees Fahrenheit to a temperature expressed in degrees Celsius. Unlike the example in the previous section, this one includes additional statements to process a special SOAP header. If you have already worked through the example in the previous section, change the CREATE below to ALTER since you are now going to modify the stored procedure.

   ```
   CREATE PROCEDURE FToCConvertor( 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 = '//*:/*:userName';
   SET namespace = SOAP_HEADER( hd_key, 1,
   ```
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 request 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 SQL code that does the SOAP header retrieval in the example is this. It exits the loop when NULL is finally returned.

```
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="SecretAgent" mustUnderstand="1">
   <userName alias="99">
```

For this header, the @namespace attribute value would be: SecretAgent

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 ( alias LONG VARCHAR '@*:alias',
         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:

<table>
<thead>
<tr>
<th>alias</th>
<th>first_name</th>
<th>last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>Susan</td>
<td>Hilton</td>
</tr>
</tbody>
</table>

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. You have the temperature in Fahrenheit degrees and you have some additional attributes that were passed to the web service in a SOAP header. So what could you do with this information?

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 next step in the stored procedure is to create a response in the SOAP format. You can build the XML response as follows:

```
SET authinfo =
    XMLELEMENT( 'Authentication',
        XMLATTRIBUTES( namespace as xmlns,
                       alias, mustUnderstand ),
        XMLELEMENT( 'first', first_name ),
        XMLELEMENT( 'last', last_name ) );
```

This builds the following XML string:

```
<Authentication xmlns="SecretAgent" alias="99"
    mustUnderstand="1">
    <first>Susan</first>
    <last>Hilton</last>
</Authentication>
```

Finally, to return the SOAP response to the caller, the SA_SET_SOAP_HEADER stored procedure is used:

```
CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
```
As in the example in the previous section, the last step is the calculation that converts from degrees Fahrenheit to degrees Celsius.

At this point, you now have a SQL Anywhere web service server running that can convert temperatures from degrees Fahrenheit to degrees Celsius as in the previous section. The major difference, however, is that it can also process a SOAP header from the caller and send a SOAP response back to the caller.

This is only half of the picture. The next step is to develop an example client that can send SOAP requests and receive SOAP responses.

If you have followed the steps outlined in the previous section, you can skip steps 1 through 3 and go directly to step 4.

**To send and receive SOAP headers**

1. Create another database for use with a second server.
   ```
   dbinit ftc_client
   ```
2. Start the personal server using this database.
   ```
   dbeng11 ftc_client.db
   ```
3. Connect to the personal server using another instance of Interactive SQL.
   ```
   dbisql -c "UID=DBA;PWD=sql;ENG=ftc_client"
   ```
4. Using Interactive SQL, create a stored procedure.
   ```
   CREATE PROCEDURE FtoC( temperature FLOAT,
               INOUT inoutheader LONG VARCHAR,
               IN inheader LONG VARCHAR )
   URL 'http://localhost:8082/FtoCService'
   TYPE 'SOAP:DOC'
   SOAPHEADER '!inoutheader!inheader';
   ```

   The URL clause is used to reference the SOAP web service. The string 'http://localhost:8082/FtoCService' specifies the URI of the web service that is going to be used. This is a reference to the web server that is listening on port 8082.

   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 alphanumeric. 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 will have to be used to extract the SOAP response header information.

5. You need a wrapper for the FtoC stored procedure so create a second stored procedure as follows. Unlike the example in the previous section, this one includes additional statements to create a special SOAP header, send it in a web service call, and process a response from the web server. If you have already worked through the example in the previous section, change the CREATE below to ALTER since you are now going to modify the stored procedure.
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;

This stored procedure acts as a cover procedure for the call to the web service. The stored procedure has been enhanced from the example in the previous section. It creates two SOAP headers. The first one is this.

    <Authentication xmlns="SecretAgent"
        mustUnderstand="1">
        <userName alias="99">'
        <first>Susan</first>
        <last>Hilton</last>
        </username>
    </Authentication>

The second one is this.

    <Session xmlns="SomeSession">123456789</Session>

When the cursor is opened, the SOAP request is sent to the web service.

    <Authentication xmlns="SecretAgent" alias="99"
            mustUnderstand="1">
        <first>Susan</first>
        <last>Hilton</last>
    </Authentication>

The FtoC stored procedure returns a result set that this stored procedure will process. The result set will look something like this.

    <tns:rowset xmlns:tns="http://localhost/ftc/FtoCService">
        <tns:row>
            <tns:answer>100</tns:answer>
        </tns:row>
    </tns:rowset>
The OPENXML function is used to parse the XML that is returned, extracting the value that is the temperature in degrees Celsius.

6. Call the stored procedure to send the request and obtain the response:

```sql
CALL FahrenheitToCelsius(212);
```

The Fahrenheit temperature and the Celsius equivalent appears.

<table>
<thead>
<tr>
<th>temperature</th>
<th>Celsius</th>
</tr>
</thead>
<tbody>
<tr>
<td>212.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

A SQL Anywhere web service client can be declared as either a function or a procedure. A SQL Anywhere client function declaration effectively restricts all parameters to in mode only (parameters cannot be assigned by the called function). Calling a SQL Anywhere web service function will return the raw SOAP envelope response whereas a procedure returns a result set.

A SOAPHEADER clause has been added to the create/alter procedure/function statements. A SOAP header can be declared as a static constant or can be dynamically set using the parameter substitution mechanism. A web service client function can define one or more in mode substitution parameters whereas a web service client procedure can also define a single inout or out substitution parameter. Therefore a web service client procedure can return the response SOAP header within an out (or inout) substitution parameter. A web service function must parse the response SOAP envelope to obtain the header entries.

The following example illustrates how a client can specify the sending of several header entries with parameters and receiving the response SOAP header data.

```sql
CREATE PROCEDURE SoapClient(
    INOUT hd1 VARCHAR,
    IN hd2 VARCHAR,
    IN hd3 VARCHAR
) URL 'localhost/some_endpoint'
SOAPHEADER '!hd1!hd2!hd3';
```

Notes

- hd1, hd2, and hd3 all specify request header entries. hd1 also returns the aggregate of all response header entries.
- When the SOAP client is called with a SOAP header, it will generate the enclosing SOAP header element. If the SOAPHEADER value is NULL then no SOAP header element is generated.
- If no SOAP header is received, then hd1 is set to NULL.
- The INOUT mode specification of hd1 is redundant since the default mode of a parameter is INOUT.
- The following runtime error results if more than one substitution parameter is specified as an OUT (or INOUT) type: 'Expression has unsupported data type' SQLCODE=-624, ODBC 3 State="HY000"
- Only a single substitution parameter explicitly used for a SOAPHEADER can be declared as OUT.

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.
Working with MIME types

The TYPE clause for a SQL Anywhere web service client procedure or function definition allows the specification of a MIME type. The value of the MIME type specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service stored procedure (or function) call after parameter substitutions have been processed. Calling a web service procedure with a null or no parameter (after substitutions) will result in a request with no body and a content-length of zero. The behavior has not changed if a MIME type is not specified. Parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

Some typical MIME types include:

- text/plain
- text/html
- text/xml

The following steps illustrate the setting of a MIME type. The first part sets up a web service that can be used to test the setting of MIME type. The second part demonstrates how to set a MIME type.

Create a web service server

1. Create a database.
   
   `dbinit echo`

2. Start a server using this database.
   
   `dbsrv11 -xs http(port=8082) -n echo echo.db`

3. Connect to the server using Interactive SQL.
   
   `dbisql -c "UID=DBA;PWD=sql;ENG=echo"`

4. Using Interactive SQL, create a web service.
   
   ```
   CREATE SERVICE EchoService
   TYPE 'RAW'
   USER DBA
   AUTHORIZATION OFF
   SECURE OFF
   AS CALL Echo(:valueAsXML);
   ```

5. Define the stored procedure that this service is to call.
   
   ```
   CREATE PROCEDURE Echo( parm LONG VARCHAR )
   BEGIN
     SELECT parm;
   END;
   ```

At this point, you now have a SQL Anywhere web service server running and ready to handle requests. The server is listening for HTTP requests on port 8082.

To use this web server for testing, create another SQL Anywhere database, start it, and connect to it. The following steps show how to do this.
To send an HTTP request

1. Using the database creation utility, create another database for use with a web service client.

   \texttt{dbinit echo_client}

2. Continuing with Interactive SQL, start this database using the following statement.

   \texttt{START DATABASE 'echo_client.db'
AS echo_client;}

3. Now, connect to the database that has been started on the server echo using the following statement.

   \texttt{CONNECT TO 'echo'
DATABASE 'echo_client'
USER 'DBA'
IDENTIFIED BY 'sql';}

4. Create a stored procedure that will communicate with the EchoService web service.

   \begin{verbatim}
   CREATE PROCEDURE setMIME(
     value LONG VARCHAR,
     mimeType LONG VARCHAR,
     urlSpec LONG VARCHAR
   )
   URL '!urlSpec'
   HEADER 'ASA-Id'
   TYPE 'HTTP:POST:!mimeType';
   \end{verbatim}

   The URL clause is used to reference the web service. For illustration purposes, the URL will be passed as a parameter to the setMIME procedure.

   The TYPE clause indicates that the MIME type will be passed as a parameter to the setMIME procedure.

   The default format used when making a web service request is 'SOAP:RPC'. The format chosen for making this web service request is 'HTTP:POST'.

5. Call the stored procedure to send the request and obtain the response. The value parameter that is passed is a URL-encoded form of \texttt{<hello>this is xml</hello>}. The media type is \texttt{application/x-www-form-urlencoded} since form-urlencoded is understood by the SQL Anywhere web server.

   The URL for the web service is included as the final parameter in the call.

   \begin{verbatim}
   CALL setMIME('valueAsXML=%3Chello%3Ethis%20is%20xml%3C/hello%3E',
   'application/x-www-form-urlencoded',
   'http://localhost:8082/EchoService');
   \end{verbatim}

   The final parameter specifies the URI of the web service that is listening on port 8082.

   The following is representative of the HTTP packet that is sent to the web server.

   \begin{verbatim}
   POST /EchoService HTTP/1.0
   Date: Sun, 28 Jan 2007 04:04:44 GMT
   Host: localhost
   Accept-Charset: windows-1252, UTF-8, *
   User-Agent: SQLAnywhere/11.0.0.1297
   Content-Type: application/x-www-form-urlencoded; charset=windows-1252
   Content-Length: 49
   ASA-Id: 1055532613:echo_client:echo:968000
   Connection: close
   valueAsXML=%3Chello%3Ethis%20is%20xml%3C/hello%3E
   \end{verbatim}
The following is the response from the web server.

HTTP/1.1 200 OK
Server: SQLAnywhere/11.0.0.1297
Date: Sun, 28 Jan 2007 04:04:44 GMT
Expires: Sun, 28 Jan 2007 04:04:44 GMT
Content-Type: text/plain; charset=windows-1252
Connection: close

<hello>this is xml</hello>

The result set that is displayed by Interactive SQL is shown next.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>Body</td>
<td>&lt;hello&gt;this is xml&lt;/hello&gt;</td>
</tr>
<tr>
<td>Server</td>
<td>SQLAnywhere/11.0.0.1297</td>
</tr>
<tr>
<td>Date</td>
<td>Sun, 16 Dec 2007 04:04:44 GMT</td>
</tr>
<tr>
<td>Expires</td>
<td>Sun, 16 Dec 2007 04:04:44 GMT</td>
</tr>
<tr>
<td>Content-Type</td>
<td>text/plain; charset=windows-1252</td>
</tr>
<tr>
<td>Connection</td>
<td>close</td>
</tr>
</tbody>
</table>
Using HTTP sessions

HTTP connections can create an HTTP session to maintain state between HTTP requests.

An **HTTP session** provides the means for persisting the client (typically a web-browser) state with minimal SQL application code. A database connection under a session context is held for the duration of the session's lifetime. Each new HTTP request that is marked with a session ID is serialized (queued) so that each request with the same session ID is sequentially processed using the same database connection. Reusing the database connection provides the means of maintaining state information between HTTP requests. In contrast, sessionless HTTP requests create a new database connection for each request and data from temporary tables and connection variables cannot be shared across requests.

HTTP session management provides support for both URL and cookie state management techniques.

A working example of the HTTP session features is provided in `samples-dir\SQLAnywhere\HTTP\session.sql`.

Creating an HTTP session

A session is created within a web application using the HTTP option SessionID by calling the `sa_set_http_option` system procedure. A session ID can be any non-null string. Internally, a session key composed of the session ID and database name is generated so that the session key is unique across databases in the event that multiple databases are loaded. The entire session key is limited to 128 characters in length. In this example, a unique session ID is generated and passed to `sa_set_http_option`.

```sql
DECLARE session_id VARCHAR(64);
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);
```

A web application can obtain the session ID through the SessionID connection property. This property is an empty string if no session ID is defined for the connection (that is, the connection is sessionless).

```sql
DECLARE session_id VARCHAR(64);
SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
```

Once the session is created with the `sa_set_http_option` procedure, a localhost client can access the session with the specified session ID (for example, `session_63315422814117`) running within the database `dbname` running the service `session_service` with the following URL.

```
http://localhost/dbname/session_service?sessionid=session_63315422814117
```

If only one database is connected, then the database name can be omitted.

```
http://localhost/session_service?sessionid=session_63315422814117
```
Session management with cookies

Cookie state management is supported using the `sa_set_http_header` system procedure with 'Set-Cookie' as the field name. Utilizing cookies for state management negates the need to include the session ID within the URL. Instead, the client provides the session ID within its HTTP cookie header. The downside to using cookies for state management is that cookie support cannot be depended upon in an unregulated environment where clients may have disabled cookies. So, a web application should support both URL and cookie session state management. A URL session ID, as described in the previous section, takes precedence in the event that a client provides both a URL and cookie session ID. It is the web application's responsibility to delete the SessionID cookie in the event that the session expires or that the session is explicitly deleted (for example, `sa_set_http_option('SessionID', NULL)`).

```sql
DECLARE session_id VARCHAR(64);
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;' );
```

Detection of stale sessions

The SessionID and SessionCreateTime connection properties are useful for determining whether the current connection is within a session context. If either connection property query returns an empty string, then the session does not exist. The SessionCreateTime property provides a metric of when a given session was created. It is immediately defined when the `sa_set_http_option` call is made. The SessionLastTime property provides the time that the session was last used. More specifically, it is the time when the last processed request for the session released the database connection upon termination of that previous request. The SessionLastTime connection property is returned as an empty string when the session is first created until the request (that created the session) releases the connection.

Deleting or changing the session ID

The session ID can be reset to another value by calling the `sa_set_http_option` system procedure with a new SessionID value. Changing the session ID has the effect of deleting the old session and creating a new session, but it reuses the current database connection so that state information is not lost. A SessionID can be set to NULL (or the empty string) which deletes the session. A SessionID cannot be set to an ID of an existing session (other than its own session ID in which case nothing happens). Trying to set a SessionID to an existing session's session ID will result in an "Invalid setting for HTTP option 'SessionID' SQLCODE=-939" error.

A server receiving a burst of multiple HTTP requests specifying the same session context queues (serializes) the requests on its session queue. In the event that the SessionID is changed or deleted by one (or more) of the requests, any pending requests in the session queue are requeued as individual HTTP requests. Each HTTP request will fail to obtain the session because it no longer exists. An HTTP request failing to obtain a session will default to sessionless operation and create a new database connection. The web application
can verify that a request is operating within a session context by checking for a non-empty string value for the SessionID or SessionCreateTime connection properties. Of course, the web application can check the state of any application specific variables or temporary tables that it uses. Here is an example:

```sql
IF VAREXISTS( 'state' ) = 0 THEN
    // first invocation by this connection
    CREATE VARIABLE state LONG VARCHAR;
END IF;
```

### Session semantics

An HTTP request can create an HTTP session context. A session created by the request is always immediately instantiated so that any subsequent HTTP requests requiring that session context is queued by the session.

An HTTP request that begins with no session but has created its session context is the creator of its session. The creator request can change or delete its session context where any change or deletion occurs immediately. An HTTP request that begins within a session context can also change or delete its session. A change made to its session immediately creates a pending session that is fully functional with the exception that another HTTP request cannot take ownership (an incoming request requiring the pending session would instead be queued on the session). To summarize, changing or deleting a session of a creator request modifies the current session context immediately, while a request only changing its session modifies its pending session. When an HTTP request finishes, it checks to see if it has a pending session. If a pending session exists, it deletes its current session and replaces it with the pending session. The database connection cached by the session is effectively moved to the new session context and all state data, such as temporary tables and created variables, are preserved.

In all cases, whenever an HTTP session is deleted, any requests within its queue are released and allowed to execute without a session context. Application code expecting that a request is running within a session context must attempt to acquire a valid session context by calling CONNECTION_PROPERTY('SessionID').

```sql
DECLARE ses_id LONG VARCHAR;
SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO ses_id;
```

If an HTTP request is canceled either intentionally or because of network failure, an existing pending session is deleted preserving the original session context. A creator HTTP request, whether canceled or having terminated normally, changes session state immediately.

### Dropping a connection and server shutdown

Explicitly dropping a database connection that is cached within a session context causes the session to be deleted. Deletion of the session in this manner is considered a cancel operation, and any HTTP requests released from the session queue are in a canceled state. This ensures that the HTTP requests are terminated quickly and signal the user appropriately.

Similarly, a server or database shutdown cancels its appropriate database connections possibly causing canceled HTTP requests.
Session timeout

The http_session_timeout public database option provides variable session timeout control. The option setting is in units of minutes. By default the public setting is 30 minutes. The minimum value is 1 minute and the maximum value is 525600 minutes (365 days). Web applications can change the timeout criteria from within any request that owns the session. A new timeout criteria may impact subsequent requests queued should the session timeout. It is up to the web application to provide the logic to detect if a client is attempting to access a non-existent session. It can do this by examining the SessionCreateTime connection property to determine if it is a valid timestamp. The SessionCreateTime value will be an empty string if the HTTP request is not associated with an existing session.

Session scope

Sessions are not persisted across server restarts.

Licensing

Since the connection associated with a session maintains its hold on a database connection for the life of the connection, it also holds one license seat as well. Web applications should take this into consideration and therefore ensure that stale sessions are deleted appropriately or have an appropriate timeout value set.

For more information about licensing in SQL Anywhere, visit http://www.sybase.com/detail?id=1056242.

Session errors

The error 503 Service Unavailable 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 error 403 Forbidden occurs if the client IP address or hostname 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.

Summary of session connection properties and options

Connection properties

- SessionID

```
SELECT CONNECTION_PROPERTY('SessionID') INTO ses_id;
```

Provides the current session ID within the current database context.
• **SessionCreateTime**

```
SELECT CONNECTION_PROPERTY('SessionCreateTime') INTO ses_create;
```

Provides the timestamp of when the session was created.

• **SessionLastTime**

```
SELECT CONNECTION_PROPERTY('SessionLastTime') INTO ses_last;
```

Provides timestamp of when the session was released by the last request.

• **http_session_timeout**

```
SELECT CONNECTION_PROPERTY('http_session_timeout') INTO ses_timeout;
```

Fetches the current session timeout in units of minutes.

**HTTP options**

• **'SessionID','value'**

```
CALL sa_set_http_option('SessionID', 'my_app_session_1');
```

Create or change a session context for the current HTTP request. Returns an error if my_app_session_1 is owned by another HTTP request.

• **'SessionID', NULL**

```
CALL sa_set_http_option('SessionID', NULL);
```

If the request comes from the session creator, the current session is deleted immediately; otherwise, the session is marked for deletion. Deleting a session when the request has no session is not an error and has no effect.

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.

Changing a session with a pending session back to its original SessionID results in the pending session being deleted.

**HTTP session timeout**

• **http_session_timeout**

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

Sets the current HTTP session timeout (in minutes). The default is 30 and the range is 1 to 525600 minutes (365 days). See “http_session_timeout option [database]” [*SQL Anywhere Server - Database Administration*].
A web application may require a means by which it can track active session usage within the database 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 code demonstrates this approach:

```sql
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 '
';
```

If you examine the database server messages window, you might see output similar to the following.

```
conn_id = 30
conn_id = 29, SessionID = session_6331544223323
conn_id = 28, SessionID = session_6331544220088
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 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 requeued as discussed in “Deleting or changing the session ID” on page 936. In the event the connection is currently inactive, the session is marked for deletion and requeued 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.

When stopping a database unconditionally, each database connection is dropped, causing all sessions under that database context to be deleted. This is guaranteed since one valid database connection must exist for one session context and a database connection can only be associated with one session at a time. Both the session and database connection must be within the same database context.

For more information about sessions in a database context, see the description of session key in “Creating an HTTP session” on page 935.
Using automatic character set conversion

By default, character set conversion is performed automatically on outgoing result sets of type text. Result sets of other types, such as binary objects, are not translated. The character set of the request is converted to the database character set, and the result set is converted from the database character set to the client character set, as required, except on binary columns in the result set. When the request lists multiple character sets that it can handle, the server takes the first suitable one from the list.

Character-set conversion can be enabled or disabled by setting the HTTP option CharsetConversion. The allowed values are ON and OFF. The default value is ON. The following statement turns automatic character-set conversion off:

```sql
CALL sa_set_http_option( 'CharsetConversion', 'OFF' );
```

For more information about built-in stored procedures, see “Alphabetical list of system procedures” [SQL Anywhere Server - SQL Reference].
Handling errors

When web service requests fail, the database server generates standard errors that appear in your browser. These errors are assigned numbers consistent with the protocol standards.

If the service is a SOAP service, faults are returned to the client as SOAP faults as defined in the SOAP version 1.1 standard:

- 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.

If the client cannot be identified as a SOAP client, then the appropriate HTTP error is returned in a generated HTML document.

The following are some of the typical errors that you may encounter:

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>SOAP fault</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>301</td>
<td>Moved permanently</td>
<td>Server</td>
<td>The requested page has been permanently moved. The server automatically redirects the request to the new location.</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
<td>Server</td>
<td>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.</td>
</tr>
<tr>
<td>307</td>
<td>Temporary Redirect</td>
<td>Server</td>
<td>The requested page has been moved, but this change may not be permanent. The server automatically redirects the request to the new location.</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>Client.BadRequest</td>
<td>The HTTP request is incomplete or malformed.</td>
</tr>
<tr>
<td>401</td>
<td>Authorization Required</td>
<td>Client.Authorization</td>
<td>Authorization is required to use the service, but a valid user name and password were not supplied.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>Client.Forbidden</td>
<td>You do not have permission to access the database.</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>SOAP fault</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>Client.NotFound</td>
<td>The named database is not running on the server, or the named web service does not exist.</td>
</tr>
<tr>
<td>408</td>
<td>Request Timeout</td>
<td>Server.RequestTimeout</td>
<td>The maximum connection idle time was exceeded while receiving the request.</td>
</tr>
<tr>
<td>411</td>
<td>HTTP Length Required</td>
<td>Client.LengthRequired</td>
<td>The server requires that the client include a Content-Length specification in the request. This typically occurs when uploading data to the server.</td>
</tr>
<tr>
<td>413</td>
<td>Entity Too Large</td>
<td>Server</td>
<td>The request exceeds the maximum permitted size.</td>
</tr>
<tr>
<td>414</td>
<td>URI Too Large</td>
<td>Server</td>
<td>The length of the URI exceeds the maximum allowed length.</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
<td>Server</td>
<td>An internal error occurred. The request could not be processed.</td>
</tr>
<tr>
<td>501</td>
<td>Not Implemented</td>
<td>Server</td>
<td>The HTTP request method is not GET, HEAD, or POST.</td>
</tr>
<tr>
<td>502</td>
<td>Bad Gateway</td>
<td>Server</td>
<td>The document requested resides on a third-party server and the server received an error from the third-party server.</td>
</tr>
<tr>
<td>503</td>
<td>Service Unavailable</td>
<td>Server</td>
<td>The number of connections exceeds the allowed maximum.</td>
</tr>
</tbody>
</table>
SQL Anywhere Database Tools Interface

This section describes the database tools programming interface for SQL Anywhere.

Database tools interface .......................................................................................................... 947
Exit codes ............................................................................................................................ 1013
Database tools interface

Contents

Introduction to the database tools interface ............................................................... 948
Using the database tools interface ............................................................................ 950
DBTools functions ..................................................................................................... 957
DBTools structures .................................................................................................... 967
DBTools enumeration types .................................................................................... 1007
**Introduction to the database tools interface**

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 `dbtool11.dll`. For Linux and Unix, the name of this library is `libdbtool11_r.so`. For Mac OS X, the name of this library is `libdbtool11_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 chapter describes the interface to the database tools library. This chapter 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 `dbtool11.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.h` header file included with SQL Anywhere 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.
- **dbtvers.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.
Using the database tools interface

This section provides an overview of how to develop applications that use the DBTools interface for managing databases.

Using the 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 libdbtool11.so (non-threaded) or libdbtool11_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:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>dbtlstm.lib</td>
</tr>
<tr>
<td>Microsoft Windows Mobile</td>
<td>dbtool11.lib</td>
</tr>
</tbody>
</table>

Starting and finishing 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 the SQL Anywhere message library. The messages 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 clean up DBTools:

```c
// 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
```
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.

```c
// 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 );
```

For information about the members of the DBTools structures, see “DBTools structures” on page 967.

Using 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:

```c
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 use this version member to hold the version of the DBTools library that your application was developed against. The current version of the DBTools library is defined when you include the `dbtools.h` header file.

To assign the current version number to a structure

- Assign the version constant to the version member of the structure before calling the DBTools function. The following line assigns the current version to a backup structure:

```c
backup_info.version = DB_TOOLS_VERSION NUMBER;
```

Compatibility

The version number allows your application to continue working against 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. If the functionality of your application doesn't change, then you may want to use one of the version-specific macros defined in `dbtoolsvers.h`, so that a library version mismatch does not occur.

Using 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:

```c
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 samples-dir\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.

```c
#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 dbackup;
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.errortn = (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 11 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("dbtool11.dll");
if(hinst == NULL)
{
    // Failed
    return EXIT_FAIL;
}
dbackup = (DBTOOLSFUNC)GetProcAddress((HMODULE)hinst,
"_DBBackup@4");
dbtoolsinit = (DBTOOLSFUNC)GetProcAddress((HMODULE)hinst,
"_DBToolsInit@4");
dbtoolsfini = (DBTOOLSPROC)GetProcAddress((HMODULE)hinst,
"_DBToolsFinii@4");
ret_code = (*dbtoolsinit)( &dbt_info );
if(ret_code != EXIT_OKAY)
{
    return ret_code;
}
ret_code = (*dbackup)( &backup_info );
(*dbtoolsfini)( &dbt_info );
FreeLibrary( hinst );
return ret_code;
DBTools functions

DBBackup function

Backs up a database. This function is used by the dbbackup utility.

Prototype

```c
short DBBackup ( const a_backup_db * );
```

Parameters

A pointer to a structure. See “a_backup_db structure” on page 967.

Return value

A return code, as listed in “Software component exit codes” on page 1014.

Remarks

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 “BACKUP statement” [SQL Anywhere Server - SQL Reference].

DBChangeLogName function

Changes the name of the transaction log file. This function is used by the dblog utility.

Prototype

```c
short DBChangeLogName ( const a_change_log * );
```

Parameters

A pointer to a structure. See “a_change_log structure” on page 969.

Return value

A return code, as listed in “Software component exit codes” on page 1014.

Remarks

The -t option of the Transaction Log utility (dblog) changes the name of the transaction log. DBChangeLogName provides a programmatic interface to this function.

For a description of the dblog utility, see “Transaction Log utility (dblog)” [SQL Anywhere Server - Database Administration].
See also

- “ALTER DATABASE statement” [SQL Anywhere Server - SQL Reference]

**DBCreate function**

Creates a database. This function is used by the dbinit utility.

**Prototype**

```c
short DBCreate ( const a_create_db *);
```

**Parameters**

A pointer to a structure. See “a_create_db structure” on page 971.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the dbinit utility, see “Initialization utility (dbinit)” [SQL Anywhere Server - Database Administration].

See also

- “CREATE DATABASE statement” [SQL Anywhere Server - SQL Reference]

**DBCreatedVersion function**

Determines the version of SQL Anywhere that was used to create a database file, without attempting to start the database. Currently, this function only differentiates between version 10 or 11 and pre-10 databases.

**Prototype**

```c
short DBCreatedVersion ( a_db_version_info *);
```

**Parameters**

A pointer to a structure. See “a_db_version_info structure” on page 975.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

If the return code indicates success, then the created_version field of the a_db_version_info structure contains a value of type a_db_version indicating which version of SQL Anywhere created the database. For the definition of the possible values, see “a_db_version enumeration” on page 1008.

Version information is not set if a failing code is returned.
See also
- “CREATE DATABASE statement” [SQL Anywhere Server - SQL Reference]
- “a_db_version_info structure” on page 975
- “a_db_version enumeration” on page 1008

**DBErase function**

Erases a database file and/or transaction log file. This function is used by the dberase utility.

**Prototype**
```
short DBErase ( const an_erase_db * );
```

**Parameters**
A pointer to a structure. See “an_erase_db structure” on page 978.

**Return value**
A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**
For information about the Erase utility and its features, see “Erase utility (dberase)” [SQL Anywhere Server - Database Administration].

**DBInfo function**

Returns information about a database file. This function is used by the dbinfo utility.

**Prototype**
```
short DBInfo ( const a_db_info * );
```

**Parameters**
A pointer to a structure. See “a_db_info structure” on page 973.

**Return value**
A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**
For information about the Information utility and its features, see “Information utility (dbinfo)” [SQL Anywhere Server - Database Administration].

See also
- “DBInfoDump function” on page 960
- “DBInfoFree function” on page 960
- “DB_PROPERTY function [System]” [SQL Anywhere Server - SQL Reference]
**DBInfoDump function**

Returns information about a database file. This function is used by the dbinfo utility when the -u option is used.

**Prototype**

```c
short DBInfoDump ( const a_db_info * );
```

**Parameters**

A pointer to a structure. See “a_db_info structure” on page 973.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the Information utility and its features, see “Information utility (dbinfo)” [SQL Anywhere Server - Database Administration].

**See also**

- “DBInfo function” on page 959
- “DBInfoFree function” on page 960
- “sa_table_page_usage system procedure” [SQL Anywhere Server - SQL Reference]

**DBInfoFree function**

Frees resources after the DBInfoDump function is called.

**Prototype**

```c
short DBInfoFree ( const a_db_info * );
```

**Parameters**

A pointer to a structure. See “a_db_info structure” on page 973.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the Information utility and its features, see “Information utility (dbinfo)” [SQL Anywhere Server - Database Administration].

**See also**

- “DBInfo function” on page 959
- “DBInfoDump function” on page 960
**DBLicense function**

Modifies or reports the licensing information of the database server.

**Prototype**

```c
short DBLicense( const a_db_lic_info * );
```

**Parameters**

A pointer to a structure. See “a_dblic_info structure” on page 976.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the Server Licensing utility and its features, see “Server Licensing utility (dblic)” [SQL Anywhere Server - Database Administration].

---

**DBRemoteSQL function**

Accesses the SQL Remote Message Agent.

**Prototype**

```c
short DBRemoteSQL( const a_remote_sql * );
```

**Parameters**

A pointer to a structure. See “a_remote_sql structure” on page 979.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the features you can access, see “Message Agent (dbremote)” [SQL Remote].

**See also**

- “SQL Remote introduction” [SQL Remote]

---

**DBSynchronizeLog function**

Synchronize a database with a MobiLink server.

**Prototype**

```c
short DBSynchronizeLog( const a_sync_db * );
```

Parameters
A pointer to a structure. See “a_sync_db structure” on page 985.

Return value
A return code, as listed in “Software component exit codes” on page 1014.

Remarks
For information about the features you can access, see “Initiating synchronization” [MobiLink - Client Administration].

See also
● “DBTools interface for dbmlsync” [MobiLink - Client Administration]

DBToolsFini function
Decrements the counter and frees resources when an application is finished with the DBTools library.

Prototype
short DBToolsFini ( const a_dbtools_info * );

Parameters
A pointer to a structure. See “a_dbtools_info structure” on page 977.

Return value
A return code, as listed in “Software component exit codes” on page 1014.

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
● “DBToolsInit function” on page 962

DBToolsInit function
Prepares the DBTools library for use.

Prototype
short DBToolsInit( const a_dbtools_info * );

Parameters
A pointer to a structure. See “a_dbtools_info structure” on page 977.
Return value
A return code, as listed in “Software component exit codes” on page 1014.

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 that DBTools uses internally.

The DBToolsInit function must be called at the start of any application that uses the DBTools interface, before any other DBTools functions. For an example, see “A DBTools example” on page 954.

See also
● “DBToolsFini function” on page 962

DBToolsVersion function
Returns the version number of the DBTools library.

Prototype
short DBToolsVersion ( void );

Return value
A short integer indicating the version number of the DBTools library.

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 953

DBTranslateLog function
Translates a transaction log file to SQL. This function is used by the dbtran utility.

Prototype
short DBTranslateLog ( const a_translate_log * );

Parameters
A pointer to a structure. See “a_translate_log structure” on page 995.

Return value
A return code, as listed in “Software component exit codes” on page 1014.
Remarks
For information about the Log Translation utility, see “Log Translation utility (dbtran)” [SQL Anywhere Server - Database Administration].

**DBTruncateLog function**
Truncates a transaction log file. This function is used by the dbbackup utility.

**Prototype**
```c
short DBTruncateLog ( const a_truncate_log * );
```

**Parameters**
A pointer to a structure. See “a_truncate_log structure” on page 999.

**Return value**
A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**
For information about the Backup utility, see “Backup utility (dbbackup)” [SQL Anywhere Server - Database Administration].

See also
- “BACKUP statement” [SQL Anywhere Server - SQL Reference]

**DBUnload function**
Unloads a database. This function is used by the dbunload and dbxtract utilities.

**Prototype**
```c
short DBUnload ( const an_unload_db * );
```

**Parameters**
A pointer to a structure. See “an_unload_db structure” on page 1000.

**Return value**
A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**
For information about the Unload utility, see “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration].

For information about the Extraction utility, see “Extraction utility (dbxtract)” [SQL Remote].
**DBUpgrade function**

Upgrades a database file. This function is used by the dbupgrad utility.

**Prototype**

```c
short DBUpgrade ( const an_upgrade_db * );
```

**Parameters**

A pointer to a structure. See “an_upgrade_db structure” on page 1004.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the Upgrade utility, see “Upgrade utility (dbupgrad)” [SQL Anywhere Server - Database Administration].

**See also**

- “ALTER DATABASE statement” [SQL Anywhere Server - SQL Reference]

---

**DBValidate function**

Validates all or part of a database. This function is used by the dbvalid utility.

**Prototype**

```c
short DBValidate ( const a_validate_db * );
```

**Parameters**

A pointer to a structure. See “a_validate_db structure” on page 1005.

**Return value**

A return code, as listed in “Software component exit codes” on page 1014.

**Remarks**

For information about the Validation utility, see “Validation utility (dbvalid)” [SQL Anywhere Server - Database Administration].

**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

- “VALIDATE statement” [SQL Anywhere Server - SQL Reference]
- “sa_validate system procedure” [SQL Anywhere Server - SQL Reference]
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.

a_backup_db structure

Holds the information needed to perform backup tasks using the DBTools library.

Syntax

```c
typedef struct a_backup_db {
    unsigned short      version;
    const char *        output_dir;
    const char *        connectparms;
    MSG_CALLBACK        confirmrtn;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_CALLBACK        statusrtn;
    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;
    const char *        hotlog_filename;
    char                backup_interrupted;
    a_chkpt_log_type    chkpt_log_type;
    a_sql_uint32        page_blocksize;
} a_backup_db;
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>output_dir</td>
<td>Path to the output directory. For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;c:\backup&quot;</td>
</tr>
</tbody>
</table>
### Member | Description
--- | ---
connectparms | Parameters needed to connect to the database. They take the form of connection strings, such as the following:

"UID=DBA;PWD=sql;DBF=samples-dir\demo.db"

The database server would be started by the connection string START parameter. For example:

"START=d:\sqlany11\bin32\dbeng11.exe"

A full example connection string including the START parameter:

"UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

confirmrtn | Callback routine for confirming an action.
errorrtn | Callback routine for handling an error message.
msgrtn | Callback routine for handling an information message.
statusrtn | Callback routine for handling a status message.
backup_database | Back up the database file (1) or not (0).
backup_logfile | Back up the transaction log file (1) or not (0).
no_confirm | Operate with (0) or without (1) confirmation.
quiet | Operate without printing messages (1), or print messages (0).
rename_log | Rename the transaction log.
truncate_log | Delete the transaction log.
rename_local_log | Rename the local backup of the transaction log.
server_backup | When set to 1, indicates backup on server using BACKUP DATABASE. Equivalent to dbbackup -s option.
hotlog_filename | File name for the live backup file.
backup_interrupted | Indicates that the operation was interrupted.
Member | Description
--- | ---
chkpt_log_type | 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.
page_blocksize | Number of pages in data blocks. Equivalent to dbbackup -b option. If set to 0, then the default is 128.

See also
- “DBBackup function” on page 957
- “a_db_version enumeration” on page 1008
- “Using callback functions” on page 951

**a_change_log structure**

Holds the information needed to perform dblog tasks using the DBTools library.

**Syntax**

```c
typedef struct a_change_log {
    unsigned short      version;
    const char *        dbname;
    const char *        logname;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    a_bit_field         query_only              : 1;
    a_bit_field         quiet                   : 1;
    a_bit_field         change_mirrorname       : 1;
    a_bit_field         change_logname          : 1;
    a_bit_field         ignore_ltm_trunc        : 1;
    a_bit_field         ignore_remote_trunc     : 1;
    a_bit_field         set_generation_number   : 1;
    a_bit_field         ignore_dbsync_trunc     : 1;
    const char *        mirrorname;
    unsigned short      generation_number;
    char *              zap_current_offset;
    char *              zap_starting_offset;
    char *              encryption_key;
} a_change_log;
```

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>dbname</td>
<td>Database file name.</td>
</tr>
<tr>
<td>logname</td>
<td>The name of the transaction log. If set to NULL, there is no log.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>query_only</td>
<td>If 1, just display the name of the transaction log. If 0, permit changing of the log name.</td>
</tr>
<tr>
<td>quiet</td>
<td>Operate without printing messages (1), or print messages (0).</td>
</tr>
<tr>
<td>change_mirrorname</td>
<td>If 1, permit changing of the log mirror name.</td>
</tr>
<tr>
<td>change_logname</td>
<td>If 1, permit changing of the transaction log name.</td>
</tr>
<tr>
<td>ignore_ltm_trunc</td>
<td>When using the Log Transfer Manager, performs the same function as the dbcc settrunc( 'ltm', 'gen_id', n ) Replication Server function. For information about dbcc, see your Replication Server documentation.</td>
</tr>
<tr>
<td>ignore_remote_trunc</td>
<td>For SQL Remote. Resets the offset kept for the purposes of the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed.</td>
</tr>
<tr>
<td>set_generation_number</td>
<td>When using the Log Transfer Manager, used after a backup is restored to set the generation number.</td>
</tr>
<tr>
<td>ignore_dbmsync_trunc</td>
<td>When using dbmsync, resets the offset kept for the purposes of the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed.</td>
</tr>
<tr>
<td>mirrorname</td>
<td>The new name of the transaction log mirror file.</td>
</tr>
<tr>
<td>generation_number</td>
<td>The new generation number. Used together with set_generation_number.</td>
</tr>
<tr>
<td>zap_current_offset</td>
<td>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.</td>
</tr>
<tr>
<td>zap_starting_offset</td>
<td>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.</td>
</tr>
<tr>
<td>encryption_key</td>
<td>The encryption key for the database file.</td>
</tr>
</tbody>
</table>
See also

- “DBChangeLogName function” on page 957
- “Using callback functions” on page 951

**a_create_db structure**

Holds the information needed to create a database using the DBTools library.

**Syntax**

```c
typedef struct a_create_db {
    unsigned short version;
    const char *dbname;
    const char *logname;
    const char *startline;
    unsigned short page_size;
    const char *default_collation;
    const char *nchar_collation;
    const char *encoding;
    MSG_CALLBACK errorrtn;
    MSG_CALLBACK msgrtn;
    a_bit_field blank_pad : 2;
    a_bit_field respect_case : 1;
    a_bit_field encrypt : 1;
    a_bit_field avoid_view_collisions : 1;
    a_bit_field jconnect : 1;
    a_bit_field checksum : 1;
    a_bit_field encrypted_tables : 1;
    a_bit_field case_sensitivity_use_default : 1;
    char verbose;
    char accent_sensitivity;
    const char *mirrorno;
    const char *data_store_type;
    const char *encryption_key;
    const char *encryption_algorithm;
    char *dba_uid;
    char *dba_pwd;
    unsigned int db_size;
    int db_size_unit;
} a_create_db;
```

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>dbname</td>
<td>Database file name.</td>
</tr>
<tr>
<td>logname</td>
<td>New transaction log name.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>startline</td>
<td>The command line used to start the database server. For example: &quot;d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>The default start line is used if this member is NULL.</td>
</tr>
<tr>
<td></td>
<td>The following is the default START parameter: &quot;dbeng11 -gp page_size -c 10M&quot;</td>
</tr>
<tr>
<td>page_size</td>
<td>The page size of the database.</td>
</tr>
<tr>
<td>default_collation</td>
<td>The collation for the database.</td>
</tr>
<tr>
<td>nchar_collation</td>
<td>If not NULL, use to generate the NCHAR COLLATION clause with specified string.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>blank_pad</td>
<td>Must be one of NO_BLANK_PADDING or BLANK_PADDING. Treat blanks as significant in string comparisons and hold index information to reflect this.</td>
</tr>
<tr>
<td></td>
<td>See “Blank padding enumeration” on page 1007.</td>
</tr>
<tr>
<td>respect_case</td>
<td>Make string comparisons case sensitive and hold index information to reflect this.</td>
</tr>
<tr>
<td>encrypt</td>
<td>When set, generates the ENCRYPTED ON or, when encrypted_tables is also set, the ENCRYPTED TABLES ON clause.</td>
</tr>
<tr>
<td>avoid_view_collisions</td>
<td>Omit the generation of Watcom SQL compatibility views SYS.SYSCOLUMN and SYS.SYSINDEXES.</td>
</tr>
<tr>
<td>jconnect</td>
<td>Include system procedures needed for jConnect.</td>
</tr>
<tr>
<td>checksum</td>
<td>Set to 1 for ON or 0 for OFF. Generates one of CHECKSUM ON or CHECKSUM OFF clauses.</td>
</tr>
<tr>
<td>encrypted_tables</td>
<td>Set to 1 for encrypted tables. Used with encrypt, generates the ENCRYPTED TABLE ON clause instead of the ENCRYPTED ON clause.</td>
</tr>
<tr>
<td>case_sensitivity_use_default</td>
<td>If set, use the default case sensitivity for the locale. This only affects UCA. When set, do not add the CASE RESPECT clause to the CREATE DATABASE statement.</td>
</tr>
<tr>
<td>verbose</td>
<td>See “Verbosity enumeration” on page 1011.</td>
</tr>
</tbody>
</table>
### Member Description

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accent_sensitivity</td>
<td>One of y, n, or f (yes, no, french). Generates one of the ACCENT RESPECT, ACCENT IGNORE or ACCENT FRENCH clauses.</td>
</tr>
<tr>
<td>mirrorname</td>
<td>Transaction log mirror name.</td>
</tr>
<tr>
<td>data_store_type</td>
<td>Reserved. Use NULL.</td>
</tr>
<tr>
<td>encryption_key</td>
<td>The encryption key for the database file. Used with encrypt, it generates the KEY clause.</td>
</tr>
<tr>
<td>encryption_algorithm</td>
<td>The encryption algorithm (AES, AES256, AES_FIPS, or AES256_FIPS). Used with encrypt and encryption_key, it generates the ALGORITHM clause.</td>
</tr>
<tr>
<td>dba_uid</td>
<td>When not NULL, generates the DBA USER xxx clause.</td>
</tr>
<tr>
<td>dba_pwd</td>
<td>When not NULL, generates the DBA PASSWORD xxx clause.</td>
</tr>
<tr>
<td>db_size</td>
<td>When not 0, generates the DATABASE SIZE clause.</td>
</tr>
<tr>
<td>db_size_unit</td>
<td>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, 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” on page 1008.</td>
</tr>
</tbody>
</table>

**See also**

- “DBCreate function” on page 958
- “Using callback functions” on page 951

### a_db_info structure

Holds the information needed to return dbinfo information using the DBTools library.

**Syntax**

```c
typedef struct a_db_info {
    unsigned short      version;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_CALLBACK        statusrtn;
    unsigned short      dbbufsize;
    char *              dbnamebuffer;
    unsigned short      logbufsize;
    char *              lognamebuffer;
};
```
unsigned short      mirrorbufsize;  
char *              mirrornamebuffer;  
unsigned short      collationnamebufsize;  
char *              collationnamebuffer;  
const char *        connectparms;  
a_bit_field         quiet    : 1;  
a_bit_field         page_usage : 1;  
a_sysinfo           sysinfo;  
a_table_info *      totals;  
a_sql_uint32        file_size;  
a_sql_uint32        free_pages;  
a_sql_uint32        bit_map_pages;  
a_sql_uint32        other_pages;  
a_bit_field         checksum : 1;  
a_bit_field         encrypted_tables : 1;  
} a_db_info;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrttn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>statusrtn</td>
<td>Callback routine for handling a status message.</td>
</tr>
<tr>
<td>dbbufsize</td>
<td>Set the length of the database file name buffer (for example, _MAX_PATH).</td>
</tr>
<tr>
<td>dbnamebuffer</td>
<td>Set the pointer to database file name buffer.</td>
</tr>
<tr>
<td>logbufsize</td>
<td>Set the length of the transaction log file name buffer (for example, _MAX_PATH).</td>
</tr>
<tr>
<td>lognamebuffer</td>
<td>Set the pointer to the transaction log file name buffer.</td>
</tr>
<tr>
<td>mirrorbufsize</td>
<td>Set the length of the mirror file name buffer (for example, _MAX_PATH).</td>
</tr>
<tr>
<td>mirrornamebuffer</td>
<td>Set the pointer to the mirror file name buffer.</td>
</tr>
<tr>
<td>collationnamebufsize</td>
<td>Set the length of the database collation name and label buffer (the maximum size is 129 including space for the null character).</td>
</tr>
<tr>
<td>collationnamebuffer</td>
<td>Set the pointer to the database collation name and label buffer.</td>
</tr>
</tbody>
</table>
### Member Description

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectparms</td>
<td>Parameters needed to connect to the database. They take the form of connection strings, such as the following:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db&quot;</td>
</tr>
<tr>
<td></td>
<td>The database server would be started by the connection string START parameter. For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>A full example connection string including the START parameter:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>quiet</td>
<td>Operate without confirming messages.</td>
</tr>
<tr>
<td>page_usage</td>
<td>1 to report page usage statistics, otherwise 0.</td>
</tr>
<tr>
<td>sysinfo</td>
<td>a_sysinfo structure (see “a_sysinfo structure” on page 993).</td>
</tr>
<tr>
<td>totals</td>
<td>Pointer to a_table_info structure (see “a_table_info structure” on page 994).</td>
</tr>
<tr>
<td>file_size</td>
<td>Size of database file.</td>
</tr>
<tr>
<td>free_pages</td>
<td>Number of free pages.</td>
</tr>
<tr>
<td>bit_map_pages</td>
<td>Number of bitmap pages in the database.</td>
</tr>
<tr>
<td>other_pages</td>
<td>Number of pages that are not table pages, index pages, free pages, or bitmap pages.</td>
</tr>
<tr>
<td>checksum</td>
<td>Database page checksums enabled if 1, disabled if 0.</td>
</tr>
<tr>
<td>encrypted_tables</td>
<td>Encrypted tables are supported if 1, disabled if 0.</td>
</tr>
</tbody>
</table>

**See also**

- “DBInfo function” on page 959
- “Using callback functions” on page 951

**a_db_version_info structure**

Holds information regarding which version of SQL Anywhere was used to create the database.
typedef struct a_db_version_info {
    unsigned short  version;
    const char      *filename;
    a_db_version    created_version;
    MSG_CALLBACK    errorrtn;
    MSG_CALLBACK    msgrtn;
} a_db_version_info;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the database file to check.</td>
</tr>
<tr>
<td>created_version</td>
<td>Set to a value of type a_db_version indicating the server version that create the database file. See “a_db_version enumeration” on page 1008.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
</tbody>
</table>

See also
- “DBCreatedVersion function” on page 958
- “a_db_version enumeration” on page 1008
- “Using callback functions” on page 951

a_dblic_info structure

Holds information containing licensing information. You must use this information only in a manner consistent with your license agreement.

typedef struct a_dblic_info {
    unsigned short      version;
    char                *exename;
    char                *username;
    char                *compname;
    a_sql_int32         nodecount;
    a_sql_int32         conncount;
    a_license_type      type;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    a_bit_field         quiet           : 1;
    a_bit_field         query_only      : 1;
    char                *installkey;
} a_dblic_info;
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>exename</td>
<td>Name of the server executable or license file.</td>
</tr>
<tr>
<td>username</td>
<td>User name for licensing.</td>
</tr>
<tr>
<td>compname</td>
<td>Company name for licensing.</td>
</tr>
<tr>
<td>nodecount</td>
<td>Number of nodes licensed.</td>
</tr>
<tr>
<td>conncount</td>
<td>Must be 1000000L.</td>
</tr>
<tr>
<td>type</td>
<td>See lictype.h for values.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msg rtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>quiet</td>
<td>Operate without printing messages (1), or print messages (0).</td>
</tr>
<tr>
<td>query_only</td>
<td>If 1, just display the license information. If 0, permit changing the information.</td>
</tr>
<tr>
<td>installkey</td>
<td>Internal use only. Set to NULL.</td>
</tr>
</tbody>
</table>

**a_dbtools_info structure**

Holds the information needed to start and finish working with the DBTools library.

Syntax

```c
typedef struct a_dbtools_info {
    MSG_CALLBACK  errorrtn;
} a_dbtools_info;
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
</tbody>
</table>

See also

- “DBToolsFini function” on page 962
- “DBToolsInit function” on page 962
- “Using callback functions” on page 951
an_erase_db structure

Holds information needed to erase a database using the DBTools library.

Syntax

```c
typedef struct an_erase_db {
    unsigned short      version;
    const char *        dbname;
    MSG_CALLBACK        confirmrtn;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    a_bit_field         quiet : 1;
    a_bit_field         erase : 1;
    const char *        encryption_key;
} an_erase_db;
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>dbname</td>
<td>Database file name to erase.</td>
</tr>
<tr>
<td>confirmrtn</td>
<td>Callback routine for confirming an action.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>quiet</td>
<td>Operate without printing messages (1), or print messages (0).</td>
</tr>
<tr>
<td>erase</td>
<td>Erase without confirmation (1) or with confirmation (0).</td>
</tr>
<tr>
<td>encryption_key</td>
<td>The encryption key for the database file.</td>
</tr>
</tbody>
</table>

See also

- “DBErase function” on page 959
- “Using callback functions” on page 951

a_name structure

Holds a linked list of names. This is used by other structures requiring lists of names.

Syntax

```c
typedef struct a_name {
    struct a_name *next;
    char            name[1];
} a_name, * p_name;
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer to the next a_name structure in the list.</td>
</tr>
<tr>
<td>name</td>
<td>The name.</td>
</tr>
</tbody>
</table>

See also

- “a_translate_log structure” on page 995
- “a_validate_db structure” on page 1005
- “an_unload_db structure” on page 1000

a_remote_sql structure

Holds information needed for the dbremote utility using the DBTools library.

Syntax

```c
typedef struct a_remote_sql {
    short               version;
    MSG_CALLBACK        confirmrtn;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_QUEUE_CALLBACK  msgqueuertn;
    char *              connectparms;
    char *              transaction_logs;
    a_bit_field         receive : 1;
    a_bit_field         send : 1;
    a_bit_field         verbose : 1;
    a_bit_field         deleted : 1;
    a_bit_field         apply : 1;
    a_bit_field         batch : 1;
    a_bit_field         more : 1;
    a_bit_field         triggers : 1;
    a_bit_field         debug : 1;
    a_bit_field         rename_log : 1;
    a_bit_field         latest_backup : 1;
    a_bit_field         scan_log : 1;
    a_bit_field         link_debug : 1;
    a_bit_field         full_q_scan : 1;
    a_bit_field         no_user_interaction : 1;
    a_bit_field         unused1 : 1;
    a_sql_uint32        max_length;
    a_sql_uint32        memory;
    a_sql_uint32        frequency;
    a_sql_uint32        threads;
    a_sql_uint32        operations;
    char *              queueparms;
    char *              locale;
    a_sql_uint32        receive_delay;
    a_sql_uint32        patience_retry;
    MSG_CALLBACK        logrtn;
    a_bit_field         use_hex_offsets : 1;
    a_bit_field         use_relative_offsets : 1;
    a_bit_field         debug_page_offsets : 1;
} a_remote_sql;
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>confirmrtn</td>
<td>Pointer to a function that prints the given message and accepts a yes or no response returning TRUE if yes and FALSE if no.</td>
</tr>
<tr>
<td>errortrn</td>
<td>Pointer to a function that prints the given error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Pointer to a function that prints the given informational (non-error) message.</td>
</tr>
<tr>
<td>msgqueuertern</td>
<td>Pointer to a function that should sleep for the number of milliseconds passed to it. This function is called with 0 when DBRemoteSQL is busy, but wants to allow the upper layer to process messages. This routine should return MSGQ_SLEEP_THROUGH normally, or MSGQ_SHUTDOWN_REQUESTED to stop SQL Remote processing.</td>
</tr>
</tbody>
</table>
### Member Description

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
</table>
| connectparms | Parameters needed to connect to the database. Corresponds to the dbremote -c option. They take the form of connection strings, such as the following:  
"UID=DBA;PWD=sql;DBF=samples-dir\demo.db"  
The database server would be started by the connection string START parameter. For example:  
"START=d:\sqlany11\bin32\dbeng11.exe"  
A full example connection string including the START parameter:  
"UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe"  
For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration]. |
| transaction_logs | Pointer to a string naming the directory with offline transaction logs. Corresponds to the transaction_logs_directory argument of dbremote. |
| receive | If receive is true, messages are received. Corresponds to the dbremote -r option.  
If receive and send are both false then both are assumed true. It is recommended that you set both to false. |
| send | If send is true, messages are sent. Corresponds to the dbremote -s option.  
If receive and send are both false then both are assumed true. It is recommended that you set both to false. |
<p>| verbose | When true, extra information is printed. Corresponds to the dbremote -v option. |
| deleted | Should be set to true. If false, messages are not deleted after they are applied. Corresponds to dbremote -p option. |
| apply | Should be set to true. If false, messages are scanned, but not applied. Corresponds to dbremote -a option. |
| batch | If true, force exit after applying messages and scanning log. Same as at least one user having 'always' send time. If false, allow run mode to be determined by remote users' send times. |
| more | Should be set to true. |</p>
<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggers</td>
<td>Should be set to false in most cases; otherwise, true means DBRemoteSQL replicates trigger actions. Corresponds to the dbremote -t option.</td>
</tr>
<tr>
<td>debug</td>
<td>Include debugging output if set true.</td>
</tr>
<tr>
<td>rename_log</td>
<td>If set to true, logs are renamed and restarted.</td>
</tr>
<tr>
<td>latest_backup</td>
<td>If set to true, only process logs that are backed up. Don't send operations from a live log. Corresponds to the dbremote -u option.</td>
</tr>
<tr>
<td>scan_log</td>
<td>Reserved; set to false.</td>
</tr>
<tr>
<td>link_debug</td>
<td>If set to true, debugging is turned on for links.</td>
</tr>
<tr>
<td>full_q_scan</td>
<td>Reserved; set to false.</td>
</tr>
<tr>
<td>no_user_interaction</td>
<td>If set to true, no user interaction is requested.</td>
</tr>
<tr>
<td>max_length</td>
<td>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.</td>
</tr>
<tr>
<td>memory</td>
<td>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.</td>
</tr>
<tr>
<td>frequency</td>
<td>Set the polling frequency for incoming messages. This value should be set to the max( 1, receive_delay/60 ). See receive_delay below.</td>
</tr>
<tr>
<td>threads</td>
<td>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.</td>
</tr>
<tr>
<td>operations</td>
<td>This value is used when applying messages. Commits are ignored until DBRemoteSQL has at least this number of operations(inserts, deletes, updates) that are uncommitted. Corresponds to the dbremote -g option.</td>
</tr>
<tr>
<td>queueparms</td>
<td>Reserved; set to NULL.</td>
</tr>
<tr>
<td>locale</td>
<td>Reserved; set to NULL.</td>
</tr>
<tr>
<td>receive_delay</td>
<td>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.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>patience_retry</td>
<td>Set this to the number of polls for incoming messages that DBRemoteSQL should wait before assuming that a message it is expecting is lost. For example, if patience_retry is 3 then DBRemoteSQL tries up to three times to receive the missing message. Afterwards, it sends a resend request. The recommended value is 1. Corresponds to the dbremote -rp option.</td>
</tr>
<tr>
<td>logrtn</td>
<td>Pointer to a function that prints the given message to a log file. These messages do not need to be seen by the user.</td>
</tr>
<tr>
<td>use_hex_offsets</td>
<td>Set to true if you want log offsets to be shown in hexadecimal notation; otherwise decimal notation will be used.</td>
</tr>
<tr>
<td>use_relative_offsets</td>
<td>Set to true if you want log offsets to be displayed as relative to the start of the current log file. Set to false if you want log offsets from the beginning of time to be displayed.</td>
</tr>
<tr>
<td>debug_page_offsets</td>
<td>Reserved; set to false.</td>
</tr>
<tr>
<td>debug_dump_size</td>
<td>Reserved; set to 0.</td>
</tr>
<tr>
<td>send_delay</td>
<td>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.</td>
</tr>
<tr>
<td>resend_urgency</td>
<td>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.</td>
</tr>
<tr>
<td>include_scan_range</td>
<td>Reserved; set to NULL.</td>
</tr>
<tr>
<td>set_window_title_rtn</td>
<td>Pointer to a function that resets the title of the window (Windows only). The title could be &quot;database_name (receiving, scanning, or sending) - default_window_title&quot;.</td>
</tr>
<tr>
<td>default_window_title</td>
<td>A pointer to the default window title string.</td>
</tr>
<tr>
<td>progress_msg_rtn</td>
<td>Pointer to a function that displays a progress message.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>progress_index_rtn</td>
<td>Pointer to a function that updates the state of the progress bar. This function takes two unsigned integer arguments <code>index</code> and <code>max</code>. 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.</td>
</tr>
<tr>
<td>argv</td>
<td>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 <code>-c</code>, <code>-cq</code>, or <code>-ek</code>.</td>
</tr>
<tr>
<td>log_size</td>
<td>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 <code>-x</code> option.</td>
</tr>
<tr>
<td>encryption_key</td>
<td>Pointer to an encryption key. Corresponds to the dbremote <code>-ek</code> option.</td>
</tr>
<tr>
<td>log_file_name</td>
<td>Pointer to the name of the DBRemoteSQL output log to which the message callbacks print their output. If <code>send</code> is true, the error log is sent to the consolidated (unless this pointer is NULL).</td>
</tr>
<tr>
<td>truncate_remote_output_file</td>
<td>Set to true to cause the remote output file to be truncated rather than appended to. See below. Corresponds to the dbremote <code>-rt</code> option.</td>
</tr>
<tr>
<td>remote_output_file_name</td>
<td>Pointer to the name of the DBRemoteSQL remote output file. Corresponds to the dbremote <code>-ro</code> or <code>-rt</code> option.</td>
</tr>
<tr>
<td>warningrtn</td>
<td>Pointer to a function that prints the given warning message. If NULL, the <code>errorrtn</code> function is called instead.</td>
</tr>
<tr>
<td>mirror_logs</td>
<td>Pointer to the name of the directory containing offline mirror transaction logs. Corresponds to the dbremote <code>-ml</code> option.</td>
</tr>
</tbody>
</table>
The dbremote tool 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)
- errorttn = (address of an appropriate routine)
- msggrtn = (address of an appropriate routine)
- confirmgrtn = (address of an appropriate routine)
- msgqueuegrtn = (address of an appropriate routine)
- logrtn = (address of an appropriate routine)
- warninggrtn = (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 function” on page 961
- “DBTools interface for dbmlsync” [MobiLink - Client Administration]

**a_sync_db structure**

Holds information needed for the dbmlsync utility using the DBTools library.

**Syntax**

```c
typedef struct a_sync_db {
    unsigned short version;
    char * connectparms;
    char * publication;
    const char * offline_dir;
    char * extended_options;
    char * script_full_path;
    const char * include_scan_range;
} a_sync_db;
```
const char * raw_file;
MSG_CALLBACK confirmrtn;
MSG_CALLBACK errorrtn;
MSG_CALLBACK msgrtn;
MSG_CALLBACK logrtn;
a_sql_uint32 debug_dump_size;
a_sql_uint32 dl_insert_width;
a_bit_field verbose : 1;
a_bit_field debug : 1;
a_bit_field debug_dump_hex : 1;
a_bit_field debug_dump_char : 1;
a_bit_field debug_page_offsets : 1;
a_bit_field use_hex_offsets : 1;
a_bit_field use_relative_offsets : 1;
a_bit_field output_to_file : 1;
a_bit_field output_to_mobile_link : 1;
a_bit_field dl_use_put : 1;
a_bit_field kill_other_connections : 1;
a_bit_field retry_remote_behind : 1;
a_bit_field ignore_debug_interrupt : 1;
SET_WINDOW_TITLE_CALLBACK set_window_title_rtn;
char * default_window_title;
MSG_QUEUE_CALLBACK msgqueuertn;
MSG_CALLBACK progress_msg_rtn;
SET_PROGRESS_CALLBACK progress_index_rtn;
char ** argv;
char ** ce_argv;
a_bit_field connectparms_allocated : 1;
a_bit_field entered_dialog : 1;
a_bit_field used_dialog_allocation : 1;
a_bit_field ignore_scheduling : 1;
a_bit_field ignore_hook_errors : 1;
a_bit_field changing_pwd : 1;
a_bit_field prompt_again : 1;
a_bit_field retry_remote_ahead : 1;
a_bit_field rename_log : 1;
a_bit_field hide_conn_str : 1;
a_bit_field hide_ml_pwd : 1;
a_sql_uint32 dlg_launch_focus;
char * mlpassword;
char * new_mlpassword;
char * verify_mlpassword;
a_sql_uint32 pub_name_cnt;
char ** pub_name_list;
USAGE_CALLBACK usage_rtn;
a_sql_uint32 log_size;
a_sql_uint32 hovering_frequency;
a_bit_short ignore_hovering : 1;
a_bit_short verbose_upload : 1;
a_bit_short verbose_upload_data : 1;
a_bit_short verbose_download : 1;
a_bit_short verbose_download_data : 1;
a_bit_short autoclose : 1;
a_bit_short ping : 1;
a_bit_short unused : 9;
char * encryption_key;
a_syncpub * upload_defs;
const char * log_file_name;
char * user_name;
a_bit_short verbose_minimum : 1;
a_bit_short verbose_hook : 1;
a_bit_short verbose_row_data : 1;
a_bit_short verbose_row_cnts : 1;
a_bit_short verbose_option_info : 1;
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
</tbody>
</table>
### Member Description

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
</table>
| connectparms      | Parameters needed to connect to the database. They take the form of connection strings, such as the following:  
> "UID=DBA;PWD=sql;DBF=samples-dir\demo.db"

The database server would be started by the connection string START parameter. For example:  
> "START=d:\sqlany11\bin32\dbeng11.exe"

A full example connection string including the START parameter:  
> "UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration]. |
<p>| publication       | Deprecated; use NULL. |
| offline_dir       | Log directory, as specified on the command line after the options. |
| extended_options  | Extended options, as specified with -e. |
| script_full_path  | Deprecated; use NULL. |
| include_scan_range| Reserved; use NULL. |
| raw_file          | Reserved; use NULL. |
| confirmrtn        | Reserved; use NULL. |
| errortrn          | Function to display error messages. |
| msgrtn            | Function to write messages to the user interface and, optionally, to the log file. |
| logrtn            | Function to write messages only to the log file. |
| debug_dump_size   | Reserved; use 0. |
| dl_insert_width   | Reserved; use 0. |
| verbose           | Deprecated; use 0. |
| debug             | Reserved; use 0. |
| debug_dump_hex    | Reserved; use 0. |</p>
<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_dump_char</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>debug_page_offsets</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>use_hex_offsets</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>use_relative_offsets</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>output_to_file</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>output_to_mobile_link</td>
<td>Reserved; use 1.</td>
</tr>
<tr>
<td>dl_use_put</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>kill_other_connections</td>
<td>TRUE if -d option is specified.</td>
</tr>
<tr>
<td>retry_remote_behind</td>
<td>TRUE if -r or -rb is specified.</td>
</tr>
<tr>
<td>ignore_debug_interrupt</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>set_window_title_rtn</td>
<td>Function to call to change the title of the dbmlsync window (Windows only).</td>
</tr>
<tr>
<td>default_window_title</td>
<td>Name of the program to display in the window caption (for example, DBMLSync).</td>
</tr>
<tr>
<td>msgqueuertn</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>- MSGQ_SLEEP_THROUGH indicates that the routine slept for the requested number of milliseconds. In most cases this is the value you should return.</td>
</tr>
<tr>
<td></td>
<td>- MSGQ_SHUTDOWN_REQUESTED indicates that you would like the synchronization to terminate as soon as possible.</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td>progress_msg_rtn</td>
<td>Function to change the text in the status window, above the progress bar.</td>
</tr>
<tr>
<td>progress_index_rtn</td>
<td>Function to update the state of the progress bar.</td>
</tr>
<tr>
<td><strong>Member</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>argv</td>
<td>argv array for this run; the last element of the array must be NULL.</td>
</tr>
<tr>
<td>ce_argv</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>connectparms_allocated</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>entered_dialog</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>used_dialog_allocation</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>ignore_scheduling</td>
<td>TRUE if -is was specified.</td>
</tr>
<tr>
<td>ignore_hook_errors</td>
<td>TRUE if -eh was specified.</td>
</tr>
<tr>
<td>changing_pwd</td>
<td>TRUE if -mn was specified.</td>
</tr>
<tr>
<td>prompt_again</td>
<td>Reserved—use 0.</td>
</tr>
<tr>
<td>retry_remote_ahead</td>
<td>TRUE if -ra was specified.</td>
</tr>
<tr>
<td>rename_log</td>
<td>TRUE if -x was specified, in which case the log file is renamed and restarted.</td>
</tr>
<tr>
<td>hide_conn_str</td>
<td>TRUE unless -vc was specified.</td>
</tr>
<tr>
<td>hide_ml_pwd</td>
<td>TRUE unless -vp was specified.</td>
</tr>
<tr>
<td>dlg_launch_focus</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>mlpassword</td>
<td>MobiLink password specified with -mp; NULL otherwise.</td>
</tr>
<tr>
<td>new_mlpassword</td>
<td>New MobiLink password specified with -mn; NULL otherwise.</td>
</tr>
<tr>
<td>verify_mlpassword</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>pub_name_cnt</td>
<td>Deprecated; use 0.</td>
</tr>
<tr>
<td>pub_name_list</td>
<td>Deprecated; use NULL.</td>
</tr>
<tr>
<td>usage_rtn</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>log_size</td>
<td>Log size in bytes, as specified with -x; otherwise 0.</td>
</tr>
<tr>
<td>hovering_frequency</td>
<td>Hovering frequency in seconds; as set with -pp.</td>
</tr>
<tr>
<td>ignore-hovering</td>
<td>True if -p was specified.</td>
</tr>
<tr>
<td>verbose_upload</td>
<td>True if -vu was specified.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>verbose_upload_data</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>verbose_download</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>verbose_download_data</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>autoclose</td>
<td>TRUE if -k was specified.</td>
</tr>
<tr>
<td>ping</td>
<td>TRUE if -pi was specified.</td>
</tr>
<tr>
<td>encryption_key</td>
<td>Database key, as specified with -ek.</td>
</tr>
<tr>
<td>upload_defs</td>
<td>Linked list of publications to be uploaded together—see a_syncpub.</td>
</tr>
<tr>
<td>log_file_name</td>
<td>Database server message log file name specified with -o or -ot.</td>
</tr>
<tr>
<td>user_name</td>
<td>MobiLink user name, specified with -u.</td>
</tr>
<tr>
<td>verbose_minimum</td>
<td>TRUE if -v was specified.</td>
</tr>
<tr>
<td>verbose_hook</td>
<td>TRUE if -vs was specified.</td>
</tr>
<tr>
<td>verbose_row_data</td>
<td>TRUE if -vr was specified.</td>
</tr>
<tr>
<td>verbose_row_cnts</td>
<td>TRUE if -vn was specified.</td>
</tr>
<tr>
<td>verbose_option_info</td>
<td>TRUE if -vo was specified.</td>
</tr>
<tr>
<td>strictly_ignore_trigger_ops</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>est_upld_row_cnt</td>
<td>Estimated number of rows to upload, specified with -urc.</td>
</tr>
<tr>
<td>status_rtn</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>warningrtn</td>
<td>Function to display warning messages.</td>
</tr>
<tr>
<td>ce_reproc_argv</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>upload_only</td>
<td>True if -uo was specified.</td>
</tr>
<tr>
<td>download_only</td>
<td>TRUE if -ds was specified.</td>
</tr>
<tr>
<td>allow_schema_change</td>
<td>TRUE if -sc was specified.</td>
</tr>
<tr>
<td>dnld_gen_num</td>
<td>TRUE if -bg was specified.</td>
</tr>
<tr>
<td>apply_dnld_file</td>
<td>File specified with -ba; otherwise NULL.</td>
</tr>
<tr>
<td><strong>Member</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>create_dnld_file</td>
<td>File specified with -bc; otherwise NULL.</td>
</tr>
<tr>
<td>sync_params</td>
<td>User authentication parameters—specified with -ap.</td>
</tr>
<tr>
<td>dnld_file_extra</td>
<td>String specified with -be.</td>
</tr>
<tr>
<td>com_server</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>trans_upload</td>
<td>TRUE if -tu was specified.</td>
</tr>
<tr>
<td>continue_download</td>
<td>TRUE if -dc is specified.</td>
</tr>
<tr>
<td>dnld_read_size</td>
<td>Value specified by -drs option.</td>
</tr>
<tr>
<td>dnld_fail_len</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>upld_fail_len</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>persist_connection</td>
<td>TRUE if -pp is specified on command line.</td>
</tr>
<tr>
<td>verbose_protocol</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>no_stream_compress</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>encrypted_stream_opts</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>no_offline_logscan</td>
<td>TRUE if -do is specified</td>
</tr>
<tr>
<td>server_mode</td>
<td>TRUE if -sm specified</td>
</tr>
<tr>
<td>allowoutside_connect</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>prompt_for_encrypt_key</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>com_server_mode</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>verbose_server</td>
<td>Reserved; use 0.</td>
</tr>
<tr>
<td>server_port</td>
<td>Value from -sp option.</td>
</tr>
<tr>
<td>preload_dlls</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>sync_profile</td>
<td>Value specified with -sp option.</td>
</tr>
<tr>
<td>sync_opt</td>
<td>Reserved; use NULL.</td>
</tr>
<tr>
<td>last_upload_def</td>
<td>Reserved; use NULL.</td>
</tr>
</tbody>
</table>
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 the dbtools.h header file for additional comments.

For more information, see “dbmlsync syntax” [MobiLink - Client Administration].

**See also**

- “DBTools interface for dbmlsync” [MobiLink - Client Administration]
- “DBSynchronizeLog function” on page 961

## a_syncpub structure

Holds information needed for the dbmlsync utility.

### Syntax

```c
typedef struct a_syncpub {
    struct a_syncpub * next;
    char * pub_name;
    char * ext_opt;
    a_bit_field allocated_by_dbsync: 1;
} a_syncpub;
```

### Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_syncpub</td>
<td>Pointer to the next node in the list, NULL for the last node.</td>
</tr>
<tr>
<td>pub_name</td>
<td>Publication name(s) specified for this -n option. This is the exact string following -n on the command line.</td>
</tr>
<tr>
<td>ext_opt</td>
<td>Extended options specified using the -eu option.</td>
</tr>
<tr>
<td>allocated_by_dbsync</td>
<td>Reserved; use FALSE.</td>
</tr>
</tbody>
</table>

**See also**

- “DBTools interface for dbmlsync” [MobiLink - Client Administration]

## a_sysinfo structure

Holds information needed for dbinfo and dbunload utilities using the DBTools library.

```c
typedef struct a_sysinfo {
    a_bit_field valid_data : 1;
    a_bit_field blank_padding : 1;
    a_bit_field case_sensitivity : 1;
    a_bit_field encryption : 1;
    char default_collation[11];
} a_sysinfo;
```
typedef struct a_table_info {
    struct a_table_info *next;
    a_sql_uint32        table_id;
    a_sql_uint32        table_pages;
    a_sql_uint32        index_pages;
    a_sql_uint32        table_used;
    a_sql_uint32        index_used;
    char *              table_name;
    a_sql_uint32        table_used_pct;
    a_sql_uint32        index_used_pct;
} a_table_info;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Next table in the list.</td>
</tr>
<tr>
<td>table_id</td>
<td>ID number for this table.</td>
</tr>
<tr>
<td>table_pages</td>
<td>Number of table pages.</td>
</tr>
<tr>
<td>index_pages</td>
<td>Number of index pages.</td>
</tr>
</tbody>
</table>

See also

- “a_db_info structure” on page 973

a_table_info structure

Holds information about a table needed as part of the a_db_info structure.
### Member Description

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_used</td>
<td>Number of bytes used in table pages.</td>
</tr>
<tr>
<td>index_used</td>
<td>Number of bytes used in index pages.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>table_used_pct</td>
<td>Table space utilization as a percentage.</td>
</tr>
<tr>
<td>index_used_pct</td>
<td>Index space utilization as a percentage.</td>
</tr>
</tbody>
</table>

### See also
- “a_db_info structure” on page 973

### a_translate_log structure

Holds information needed for transaction log translation using the DBTools library.

### Syntax

```c
typedef struct a_translate_log {
    unsigned short      version;
    const char *        connectparms;
    const char *        logname;
    const char *        sqiname;
    const char *        encryption_key;
    const char *        logs_dir;
    p_name              userlist;
    a_sql_uint32        since_time;
    MSG_CALLBACK        confirmrtn;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_CALLBACK        logrtn;
    MSG_CALLBACK        statusrtn;
    char                userlisttype;
    a_bit_field         quiet                   : 1;
    a_bit_field         remove_rollback         : 1;
    a_bit_field         ansi_sql                : 1;
    a_bit_field         since_checkpoint        : 1;
    a_bit_field         replace                 : 1;
    a_bit_field         include_trigger_trans   : 1;
    a_bit_field         comment_trigger_trans   : 1;
    a_bit_field         debug                   : 1;
    a_bit_field         debug_sql_remote        : 1;
    a_bit_field         debug_dump_hex          : 1;
    a_bit_field         debug_dump_char         : 1;
    a_bit_field         debug_page_offsets      : 1;
    a_bit_field         omit_comments           : 1;
    a_bit_field         use_hex_offsets         : 1;
    a_bit_field         use_relative_offsets    : 1;
    a_bit_field         include_audit           : 1;
    a_bit_field         chronological_order     : 1;
    a_bit_field         force_recovery          : 1;
    a_bit_field         include_subsets         : 1;
} a_translate_log;
```
 Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>connectparms</td>
<td>Parameters needed to connect to the database. They take the form of connection strings, such as the following:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db&quot;</td>
</tr>
<tr>
<td></td>
<td>The database server would be started by the connection string START parameter. For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>A full example connection string including the START parameter:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>logname</td>
<td>Name of the transaction log file. If NULL, there is no log.</td>
</tr>
<tr>
<td>sqlname</td>
<td>Name of the SQL output file. If NULL, then name is based on transaction log file name (-n sets this string).</td>
</tr>
<tr>
<td>encryption_key</td>
<td>Specify database encryption key (-ek sets string).</td>
</tr>
<tr>
<td>logs_dir</td>
<td>Transaction logs directory (-m dir sets string); sqlname must be set and connect_parms must be NULL.</td>
</tr>
<tr>
<td>userlist</td>
<td>A linked list of user names. Equivalent to -u user1,... or -x user1,... Select or omit transactions for listed users.</td>
</tr>
<tr>
<td><strong>Member</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>since_time</td>
<td>Output from most recent checkpoint prior to time (-j &lt;time&gt; sets this). The number of minutes since January 1, 0001.</td>
</tr>
<tr>
<td>confirmrtn</td>
<td>Callback routine for confirming an action.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>logrtn</td>
<td>Callback routine to write messages only to the log file.</td>
</tr>
<tr>
<td>statusrtn</td>
<td>Callback routine for handling a status message.</td>
</tr>
<tr>
<td>userlisttype</td>
<td>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.</td>
</tr>
<tr>
<td>quiet</td>
<td>Set to TRUE to operate without printing messages (-y).</td>
</tr>
<tr>
<td>remove_rollback</td>
<td>Normally set to TRUE; Set to FALSE if you want to include rollback transactions in output (equivalent to -a).</td>
</tr>
<tr>
<td>ansi_sql</td>
<td>Set to TRUE if you want to produce ANSI standard SQL transactions (equivalent to -s).</td>
</tr>
<tr>
<td>since_checkpoint</td>
<td>Set to TRUE if you want output from most recent checkpoint (equivalent to -f).</td>
</tr>
<tr>
<td>replace</td>
<td>Replace existing SQL file without confirmation (equivalent to -y).</td>
</tr>
<tr>
<td>include_trigger_trans</td>
<td>Set TRUE to include trigger-generated transactions (equivalent to -g, -sr or -t).</td>
</tr>
<tr>
<td>comment_trigger_trans</td>
<td>Set TRUE to include trigger-generated transactions as comments (equivalent to -z).</td>
</tr>
<tr>
<td>debug</td>
<td>Reserved; set to FALSE.</td>
</tr>
<tr>
<td>debug_sql_remote</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>debug_dump_hex</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>debug_dump_char</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>debug_page_offsets</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>use_hex_offsets</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>use_relative_offsets</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>include_audit</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>chronological_order</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>force_recovery</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>include_subsets</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>force_chaining</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>generate_reciprocals</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>match_mode</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>show_undo</td>
<td>Reserved, use FALSE.</td>
</tr>
<tr>
<td>debug_dump_size</td>
<td>Reserved, use 0.</td>
</tr>
<tr>
<td>recovery_ops</td>
<td>Reserved, use 0.</td>
</tr>
<tr>
<td>recovery_bytes</td>
<td>Reserved, use 0.</td>
</tr>
<tr>
<td>include_source_sets</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>include_destination_sets</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>include_scan_range</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>repserver_users</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>include_tables</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>include_publications</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>queueparms</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>match_pos</td>
<td>Reserved, use NULL.</td>
</tr>
<tr>
<td>leave_output_on_error</td>
<td>Set to TRUE if you want to leave the generated .SQL file if corruption detected (equivalent to -k)</td>
</tr>
</tbody>
</table>

The members correspond to features accessible from the dbtran utility.

See the dbtools.h header file for additional comments.
See also
● “DBTranslateLog function” on page 963
● “a_name structure” on page 978
● “dbtran_userlist_type enumeration” on page 1009
● “Using callback functions” on page 951

a_truncate_log structure

Holds information needed for transaction log truncation using the DBTools library.

Syntax

typedef struct a_truncate_log {
    unsigned short version;
    const char * connectparms;
    MSG_CALLBACK errorrtn;
    MSG_CALLBACK msgrtn;
    a_bit_field quiet           : 1;
    a_bit_field server_backup   : 1;
    char        truncate_interrupted;
} a_truncate_log;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
</tbody>
</table>
| connectparms | Parameters needed to connect to the database. They take the form of connection strings, such as the following: 

"UID=DBA;PWD=sql;DBF=samples-dir\demo.db"

The database server would be started by the connection string START parameter. For example:

"START=d:\sqlany11\bin32\dbeng11.exe"

A full example connection string including the START parameter:

"UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe"

For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration]. |
<p>| errorrtn   | Callback routine for handling an error message. |
| msgrttn    | Callback routine for handling an information message. |
| quiet      | Operate without printing messages (1), or print messages (0). |</p>
<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_backup</td>
<td>When set to 1, indicates backup on server using BACKUP DATABASE. Equivalent to dbbackup -s option.</td>
</tr>
<tr>
<td>truncate_interrupted</td>
<td>Indicates that the operation was interrupted.</td>
</tr>
</tbody>
</table>

**See also**
- “DBTruncateLog function” on page 964
- “Using callback functions” on page 951

**an_unload_db structure**

Holds information needed to unload a database using the DBTools library or extract a remote database for SQL Remote. Those fields used by the dbxtract SQL Remote Extraction utility are indicated.

**Syntax**

```c
typedef struct an_unload_db {
    unsigned short      version;
    const char *        connectparms;
    const char *        temp_dir;
    const char *        reload_filename;
    char *              reload_connectparms;
    char *              reload_db_filename;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_CALLBACK        statusrtn;
    MSG_CALLBACK        confirmrtn;
    char                unload_type;
    char                verbose;
    char                escape_char;
    char                unload_interrupted;
    a_bit_field         unordered               : 1;
    a_bit_field         no_confirm              : 1;
    a_bit_field         use_internalUnload     : 1;
    a_bit_field         refresh_matView        : 1;
    a_bit_field         tableListProvided     : 1;
    a_bit_field         excludeTables         : 1;
    a_bit_field         preserveIds           : 1;
    a_bit_field         replaceDb             : 1;
    a_bit_field         escapeCharPresent     : 1;
    a_bit_field         useInternalReload     : 1;
    a_bit_field         recompute             : 1;
    a_bit_field         make Auxiliary        : 1;
    a_bit_field         encryptedTables       : 1;
    a_bit_field         removeEncryptedTables : 1;
    a_bit_field         extract               : 1;
    a_bit_field         startSubscriptions    : 1;
    a_bit_field         excludeForeignKeys    : 1;
    a_bit_field         excludeProcedures     : 1;
    a_bit_field         excludeTriggers       : 1;
    a_bit_field         excludeViews          : 1;
    a_bit_field         isolationSet          : 1;
    a_bit_field         includeWhereSubscribe : 1;
    a_bit_field         excludeHooks          : 1;
} an_unload_db;
```
Members

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>connectparms</td>
<td>Parameters needed to connect to the database. They take the form of</td>
</tr>
<tr>
<td></td>
<td>connection strings, such as the following:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db&quot;</td>
</tr>
<tr>
<td></td>
<td>The database server would be started by the connection string START</td>
</tr>
<tr>
<td></td>
<td>parameter. For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;START=d:\sqlany\bin32\dbeng.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>A full example connection string including the START parameter:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany\bin32\dbeng.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>For a list of connection parameters, see “Connection parameters” [SQL</td>
</tr>
<tr>
<td></td>
<td>Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>temp_dir</td>
<td>Directory for unloading data files.</td>
</tr>
<tr>
<td>reload_filename</td>
<td>The dbunload -r option, something like reload.sql.</td>
</tr>
<tr>
<td>reload_connectparms</td>
<td>User ID, password, database for reload database.</td>
</tr>
<tr>
<td>Members</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reload_db_filename</td>
<td>The file name of reload database to create.</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msgrtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>statusrtn</td>
<td>Callback routine for handling a status message.</td>
</tr>
<tr>
<td>confirmrtn</td>
<td>Callback routine for confirming an action.</td>
</tr>
<tr>
<td>unload_type</td>
<td>See “dbunload type enumeration” on page 1010.</td>
</tr>
<tr>
<td>verbose</td>
<td>See “Verbosity enumeration” on page 1011.</td>
</tr>
<tr>
<td>escape_char</td>
<td>Used when escape_char_present is TRUE.</td>
</tr>
<tr>
<td>unload_interrupted</td>
<td>Set if unload interrupted.</td>
</tr>
<tr>
<td>unordered</td>
<td>dbunload -u sets TRUE.</td>
</tr>
<tr>
<td>no_confirm</td>
<td>dbunload -y sets TRUE.</td>
</tr>
<tr>
<td>use_internalUnload</td>
<td>dbunload -ii/-ix sets TRUE. dbunload -xi/-xx sets FALSE.</td>
</tr>
<tr>
<td>refresh_mat_view</td>
<td>dbunload -g sets TRUE.</td>
</tr>
<tr>
<td>table_list_provided</td>
<td>dbunload -e list or -i sets TRUE.</td>
</tr>
<tr>
<td>exclude_tables</td>
<td>dbunload -e sets TRUE. dbunload -i (undocumented) sets FALSE.</td>
</tr>
<tr>
<td>preserve_ids</td>
<td>dbunload sets TRUE/-m sets FALSE.</td>
</tr>
<tr>
<td>replace_db</td>
<td>dbunload -ar sets TRUE.</td>
</tr>
<tr>
<td>escape_char_present</td>
<td>dbunload -p sets TRUE. Note that escape_char must be set.</td>
</tr>
<tr>
<td>use_internal_reload</td>
<td>Usually set TRUE; -ix/-xx sets FALSE; -ii/-xi sets TRUE.</td>
</tr>
<tr>
<td>recompute</td>
<td>dbunload -dc sets TRUE. Re-compute all computed columns.</td>
</tr>
<tr>
<td>make_auxiliary</td>
<td>dbunload -k sets TRUE. Make auxiliary catalog (for use with diagnostic tracing).</td>
</tr>
<tr>
<td>encrypted_tables</td>
<td>dbunload -et sets TRUE. Enable encrypted tables in new database (used with -an or -ar).</td>
</tr>
<tr>
<td>remove_encrypted_tables</td>
<td>dbunload -er sets TRUE. Remove encryption from encrypted tables.</td>
</tr>
<tr>
<td>Members</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extract</td>
<td>TRUE if dbxtract, otherwise FALSE.</td>
</tr>
<tr>
<td>start_subscriptions</td>
<td>dbxtract TRUE by default, -b sets FALSE.</td>
</tr>
<tr>
<td>exclude_foreign_keys</td>
<td>dbxtract -xf sets TRUE.</td>
</tr>
<tr>
<td>exclude_procedures</td>
<td>dbxtract -xp sets TRUE.</td>
</tr>
<tr>
<td>exclude_triggers</td>
<td>dbxtract -xt sets TRUE.</td>
</tr>
<tr>
<td>exclude_views</td>
<td>dbxtract -xv sets TRUE.</td>
</tr>
<tr>
<td>isolation_set</td>
<td>dbxtract -l sets TRUE.</td>
</tr>
<tr>
<td>include_where_subscribe</td>
<td>dbxtract -f sets TRUE.</td>
</tr>
<tr>
<td>exclude_hooks</td>
<td>dbxtract -hx sets TRUE.</td>
</tr>
<tr>
<td>startline_name</td>
<td>(internal use)</td>
</tr>
<tr>
<td>debug</td>
<td>(internal use)</td>
</tr>
<tr>
<td>compress_output</td>
<td>dbunload -cp sets TRUE.</td>
</tr>
<tr>
<td>schema_reload</td>
<td>(internal use)</td>
</tr>
<tr>
<td>genscript</td>
<td>(internal use)</td>
</tr>
<tr>
<td>runscript</td>
<td>(internal use)</td>
</tr>
<tr>
<td>display_create</td>
<td>-cm sets TRUE</td>
</tr>
<tr>
<td>display_create_dbinit</td>
<td>-cm dbinit sets TRUE</td>
</tr>
<tr>
<td>preserve_identity_values</td>
<td>dbunload -l sets TRUE</td>
</tr>
<tr>
<td>ms_filename</td>
<td>(internal use)</td>
</tr>
<tr>
<td>ms_reserve</td>
<td>(internal use)</td>
</tr>
<tr>
<td>ms_size</td>
<td>(internal use)</td>
</tr>
<tr>
<td>notemp_size</td>
<td>(internal use)</td>
</tr>
<tr>
<td>table_list</td>
<td>Selective table list</td>
</tr>
<tr>
<td>sysinfo</td>
<td>(internal use)</td>
</tr>
</tbody>
</table>
The members correspond to features accessible from the dbunload and dbxtract utilities.

See the dbtools.h header file for additional comments.

See also

- “DBUnload function” on page 964
- “a_name structure” on page 978
- “dbunload type enumeration” on page 1010
- “Verbosity enumeration” on page 1011
- “Using callback functions” on page 951

### an_upgrade_db structure

Holds information needed to upgrade a database using the DBTools library.

**Syntax**

```c
typedef struct an_upgrade_db {
    unsigned short      version;
    const char *        connectparms;
    MSG_CALLBACK        errorrtn;
    MSG_CALLBACK        msgrtn;
    MSG_CALLBACK        statusrtn;
    a_bit_field         quiet           : 1;
} an_upgrade_db;
```
typedef struct a_validate_db {
    unsigned short      version;
    const char *        connectparms;
    p_name               tables;
    MSG_CALLBACK        errorrtn;
    jconnect             : 1;
} a_validate_db;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
<tr>
<td>connectparms</td>
<td>Parameters needed to connect to the database. They take the form of connection strings, such as the following:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db&quot;</td>
</tr>
<tr>
<td></td>
<td>The database server would be started by the connection string START parameter. For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>A full example connection string including the START parameter:</td>
</tr>
<tr>
<td></td>
<td>&quot;UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe&quot;</td>
</tr>
<tr>
<td></td>
<td>For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>errorrtn</td>
<td>Callback routine for handling an error message.</td>
</tr>
<tr>
<td>msg rtn</td>
<td>Callback routine for handling an information message.</td>
</tr>
<tr>
<td>statusrtn</td>
<td>Callback routine for handling a status message.</td>
</tr>
<tr>
<td>quiet</td>
<td>Operate without printing messages (1), or print messages (0).</td>
</tr>
<tr>
<td>jconnect</td>
<td>Upgrade the database to include jConnect procedures.</td>
</tr>
</tbody>
</table>

See also

- “DBUpgrade function” on page 965
- “Using callback functions” on page 951

a_validate_db structure

Holds information needed for database validation using the DBTools library.

Syntax

```c
typedef struct a_validate_db {
    unsigned short version;
    const char * connectparms;
    p_name tables;
    MSG_CALLBACK errorrtn;
} a_validate_db;
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>DBTools version number.</td>
</tr>
</tbody>
</table>
| connectparms | Parameters needed to connect to the database. They take the form of connection strings, such as the following:  
"UID=DBA;PWD=sql;DBF=samples-dir\demo.db"  
The database server would be started by the connection string START parameter.  
For example:  
"START=d:\sqlany11\bin32\dbeng11.exe"  
A full example connection string including the START parameter:  
"UID=DBA;PWD=sql;DBF=samples-dir\demo.db;START=d:\sqlany11\bin32\dbeng11.exe"  
For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration]. |
| tables    | Pointer to a linked list of table names. |
| errorrtn  | Callback routine for handling an error message. |
| msgrtn    | Callback routine for handling an information message. |
| statusrtn | Callback routine for handling a status message. |
| quiet     | Operate without printing messages (1), or print messages (0). |
| index     | Validate indexes. |
| type      | See “a_validate_type enumeration” on page 1010. |

See also

- “DBValidate function” on page 965
- “a_name structure” on page 978
- “a_validate_type enumeration” on page 1010
- For more information about callback functions, see “Using callback functions” on page 951.
DBTools enumeration types

This section lists the enumeration types that are used by the DBTools library. The enumerations are listed alphabetically.

Blank padding enumeration

Used in the “a_create_db structure” on page 971, to specify the value of blank_pad.

Syntax

```c
enum {
    NO_BLANK_PADDING,
    BLANK_PADDING
};
```

Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_BLANK_PADDING</td>
<td>Does not use blank padding.</td>
</tr>
<tr>
<td>BLANK_PADDING</td>
<td>Uses blank padding.</td>
</tr>
</tbody>
</table>

See also
- “a_create_db structure” on page 971

a_chkpt_log_type enumeration

Used in the “a_backup_db structure” on page 967, to control copying of checkpoint log.

Syntax

```c
typedef enum {
    BACKUP_CHKPT_LOG_COPY = 0,
    BACKUP_CHKPT_LOG_NOCOPY,
    BACKUP_CHKPT_LOG_RECOVER,
    BACKUP_CHKPT_LOG_AUTO,
    BACKUP_CHKPT_LOG_DEFAULT
} a_chkpt_log_type;
```

Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP_CHKPT_LOG_COPY</td>
<td>Use to generate WITH CHECKPOINT LOG COPY clause.</td>
</tr>
<tr>
<td>BACKUP_CHKPT_LOG_NOCOPY</td>
<td>Use to generate WITH CHECKPOINT LOG NOCOPY clause.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BACKUP_CHKPT_LOG_RECOVER</td>
<td>Use to generate WITH CHECKPOINT LOG RECOVER clause.</td>
</tr>
<tr>
<td>BACKUP_CHKPT_LOG_AUTO</td>
<td>Use to generate WITH CHECKPOINT LOG AUTO clause.</td>
</tr>
<tr>
<td>BACKUP_CHKPT_LOG_DEFAULT</td>
<td>Use to omit WITH CHECKPOINT clause.</td>
</tr>
</tbody>
</table>

See also
- “a_backup_db structure” on page 967

### a_db_version enumeration

Used in the “a_db_version_info structure” on page 975, to indicate the version of SQL Anywhere that initially created the database.

**Syntax**
```
enum {
    VERSION_UNKNOWN,
    VERSION_PRE_10,
    VERSION_10,
    VERSION_11
};
```

**Parameters**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION_UNKNOWN</td>
<td>Unable to determine the version of SQL Anywhere that created the database.</td>
</tr>
<tr>
<td>VERSION_PRE_10</td>
<td>Database was created using a pre-10 version of SQL Anywhere.</td>
</tr>
<tr>
<td>VERSION_10</td>
<td>Database was created using SQL Anywhere 10.</td>
</tr>
<tr>
<td>VERSION_11</td>
<td>Database was created using SQL Anywhere 11.</td>
</tr>
</tbody>
</table>

See also
- “DBCreatedVersion function” on page 958
- “a_db_version_info structure” on page 975

### Database size unit enumeration

Used in the “a_create_db structure” on page 971, to specify the value of db_size_unit.
Syntax
```c
enum {
    DBSP_UNIT_NONE,
    DBSP_UNIT_PAGES,
    DBSP_UNIT_BYTES,
    DBSP_UNIT_KILOBYTES,
    DBSP_UNIT_MEGABYTES,
    DBSP_UNIT_GIGABYTES,
    DBSP_UNIT_TERABYTES
};
```

Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSP_UNIT_NONE</td>
<td>Units not specified.</td>
</tr>
<tr>
<td>DBSP_UNIT_PAGES</td>
<td>Size is specified in pages.</td>
</tr>
<tr>
<td>DBSP_UNIT_BYTES</td>
<td>Size is specified in bytes.</td>
</tr>
<tr>
<td>DBSP_UNIT_KILOBYTES</td>
<td>Size is specified in kilobytes.</td>
</tr>
<tr>
<td>DBSP_UNIT_MEGABYTES</td>
<td>Size is specified in megabytes.</td>
</tr>
<tr>
<td>DBSP_UNIT_GIGABYTES</td>
<td>Size is specified in gigabytes.</td>
</tr>
<tr>
<td>DBSP_UNIT_TERABYTES</td>
<td>Size is specified in terabytes.</td>
</tr>
</tbody>
</table>

See also

- “a_create_db structure” on page 971

**dbtran_userlist_type enumeration**

The type of a user list, as used by an “a_translate_log structure” on page 995.

Syntax
```c
typedef enum dbtran_userlist_type {
    DBTRAN_INCLUDE_ALL,
    DBTRAN_INCLUDE_SOME,
    DBTRAN_EXCLUDE_SOME
} dbtran_userlist_type;
```

Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTRAN_INCLUDE_ALL</td>
<td>Include operations from all users.</td>
</tr>
</tbody>
</table>
### dftran_value enumeration

The type of validation being performed, as used by the “a_validate_db structure” on page 1005.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTRAN_INCLUDE_SOME</td>
<td>Include operations only from the users listed in the supplied user list.</td>
</tr>
<tr>
<td>DBTRAN_EXCLUDE_SOME</td>
<td>Exclude operations from the users listed in the supplied user list.</td>
</tr>
</tbody>
</table>

**See also**
- “a_translate_log structure” on page 995

### dbunload type enumeration

The type of unload being performed, as used by the “an_unload_db structure” on page 1000.

**Syntax**

```c
enum {
    UNLOAD_ALL,
    UNLOAD_DATA_ONLY,
    UNLOAD_NO_DATA,
    UNLOAD_NO_DATA_FULL_SCRIPT
};
```

**Parameters**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNLOAD_ALL</td>
<td>Unload both data and schema.</td>
</tr>
<tr>
<td>UNLOAD_DATA_ONLY</td>
<td>Unload data. Do not unload schema. Equivalent to dbunload -d option.</td>
</tr>
<tr>
<td>UNLOAD_NO_DATA</td>
<td>No data. Unload schema only. Equivalent to dbunload -n option.</td>
</tr>
<tr>
<td>UNLOAD_NO_DATA_FULL_SCRIPT</td>
<td>No data. Include LOAD/INPUT statements in reload script. Equivalent to dbunload -nl option.</td>
</tr>
</tbody>
</table>

**See also**
- “an_unload_db structure” on page 1000

### a_validate_type enumeration

The type of validation being performed, as used by the “a_validate_db structure” on page 1005.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNLOAD_ALL</td>
<td>Unload both data and schema.</td>
</tr>
<tr>
<td>UNLOAD_DATA_ONLY</td>
<td>Unload data. Do not unload schema. Equivalent to dbunload -d option.</td>
</tr>
<tr>
<td>UNLOAD_NO_DATA</td>
<td>No data. Unload schema only. Equivalent to dbunload -n option.</td>
</tr>
<tr>
<td>UNLOAD_NO_DATA_FULL_SCRIPT</td>
<td>No data. Include LOAD/INPUT statements in reload script. Equivalent to dbunload -nl option.</td>
</tr>
</tbody>
</table>

**See also**
- “an_unload_db structure” on page 1000
### Syntax

typedef enum {
    VALIDATE_NORMAL = 0,
    VALIDATE_DATA,
    VALIDATE_INDEX,
    VALIDATE_EXPRESS,
    VALIDATE_FULL,
    VALIDATE_CHECKSUM,
    VALIDATE_DATABASE,
    VALIDATE_COMPLETE
} a_validate_type;

### Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALIDATE_NORMAL</td>
<td>Validate with the default check only.</td>
</tr>
<tr>
<td>VALIDATE_DATA</td>
<td>(obsolete)</td>
</tr>
<tr>
<td>VALIDATE_INDEX</td>
<td>(obsolete)</td>
</tr>
<tr>
<td>VALIDATE_EXPRESS</td>
<td>Validate with express check. Equivalent to dbvalid -fx option.</td>
</tr>
<tr>
<td>VALIDATE_FULL</td>
<td>(obsolete)</td>
</tr>
<tr>
<td>VALIDATE_CHECKSUM</td>
<td>Validate database checksums. Equivalent to dbvalid -s option.</td>
</tr>
<tr>
<td>VALIDATE_DATABASE</td>
<td>Validate database. Equivalent to dbvalid -d option.</td>
</tr>
<tr>
<td>VALIDATE_COMPLETE</td>
<td>Perform all possible validation activities.</td>
</tr>
</tbody>
</table>

### See also

- “a_validate_db structure” on page 1005
- “Validation utility (dbvalid)” [SQL Anywhere Server - Database Administration]
- “VALIDATE statement” [SQL Anywhere Server - SQL Reference]

### Verbosity enumeration

Specifies the volume of output.

### Syntax

enum {
    VB_QUIET,
    VB_NORMAL,
    VB_VERBOSE
};
## Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB_QUIET</td>
<td>No output.</td>
</tr>
<tr>
<td>VB_NORMAL</td>
<td>Normal amount of output.</td>
</tr>
<tr>
<td>VB_VERBOSE</td>
<td>Verbose output, useful for debugging.</td>
</tr>
</tbody>
</table>

**See also**

- “a_create_db structure” on page 971
- “an_unload_db structure” on page 1000
Exit codes

Contents

Software component exit codes ................................................................. 1014
### Software component exit codes

All database tools are provided as entry points in a DLL. These entry points use the following exit codes. The SQL Anywhere utilities (dbbackup, dbspawn, dbeng11, and so on) also use these exit codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Status</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EXIT_OKAY</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>EXIT_FAIL</td>
<td>General failure</td>
</tr>
<tr>
<td>2</td>
<td>EXIT_BAD_DATA</td>
<td>Invalid file format</td>
</tr>
<tr>
<td>3</td>
<td>EXIT_FILE_ERROR</td>
<td>File not found, unable to open</td>
</tr>
<tr>
<td>4</td>
<td>EXIT_OUT_OF_MEMORY</td>
<td>Out of memory</td>
</tr>
<tr>
<td>5</td>
<td>EXIT_BREAK</td>
<td>Terminated by the user</td>
</tr>
<tr>
<td>6</td>
<td>EXIT_COMMUNICATIONS_FAIL</td>
<td>Failed communications</td>
</tr>
<tr>
<td>7</td>
<td>EXIT_MISSING_DATABASE</td>
<td>Missing a required database name</td>
</tr>
<tr>
<td>8</td>
<td>EXIT_PROTOCOL_MISMATCH</td>
<td>Client/server protocol mismatch</td>
</tr>
<tr>
<td>9</td>
<td>EXIT_UNABLE_TO_CONNECT</td>
<td>Unable to connect to the database server</td>
</tr>
<tr>
<td>10</td>
<td>EXIT_ENGINE_NOT_RUNNING</td>
<td>Database server not running</td>
</tr>
<tr>
<td>11</td>
<td>EXIT_SERVER_NOT_FOUND</td>
<td>Database server not found</td>
</tr>
<tr>
<td>12</td>
<td>EXIT_BAD_ENCRYPT_KEY</td>
<td>Missing or bad encryption key</td>
</tr>
<tr>
<td>13</td>
<td>EXIT_DB_VER_NEWER</td>
<td>Server must be upgraded to run database</td>
</tr>
<tr>
<td>14</td>
<td>EXIT_FILE_INVALID_DB</td>
<td>File is not a database</td>
</tr>
<tr>
<td>15</td>
<td>EXIT_LOG_FILE_ERROR</td>
<td>Log file was missing or other error</td>
</tr>
<tr>
<td>16</td>
<td>EXIT_FILE_IN_USE</td>
<td>File in use</td>
</tr>
<tr>
<td>17</td>
<td>EXIT_FATAL_ERROR</td>
<td>Fatal error or assertion occurred</td>
</tr>
<tr>
<td>255</td>
<td>EXIT_USAGE</td>
<td>Invalid parameters on the command line</td>
</tr>
</tbody>
</table>

These exit codes are contained in the `install-dir\sdk\include\sqldef.h` file.
Deploying SQL Anywhere

This section introduces you to deployment strategies for SQL Anywhere.

Deploying databases and applications ................................................................. 1017
Deploying databases and applications

Contents

Introduction to deployment ................................................................. 1018
Understanding installation directories and file names ...................... 1020
Using the Deployment Wizard .......................................................... 1023
Using a silent install for deployment .................................................. 1027
Deploying client applications .......................................................... 1029
Deploying administration tools .......................................................... 1048
Deploying database servers .............................................................. 1074
Deploying external environment support ......................................... 1079
 Deploying security ........................................................................ 1081
Deploying embedded database applications .................................. 1082
Introduction to 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.

Check your license agreement
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 chapter:

- Determining required files based on the choice of application platform and architecture.
- Configuring client applications.

Much of the chapter 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 “Using the Deployment Wizard” on page 1023 and “Using a silent install for deployment” on page 1027.

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 “Deploying MobiLink applications” [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 “Using the Deployment Wizard” on page 1023 or “Using a silent install for deployment” on page 1027.

- **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 chapter 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.
Understanding 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 a full description of 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 directory structure is as follows:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/opt/sqlanywhere11/bin32</code> and <code>/opt/sqlanywhere11/bin64</code></td>
<td>Executable files, License files</td>
</tr>
<tr>
<td><code>/opt/sqlanywhere11/lib32</code> and <code>/opt/sqlanywhere11/lib64</code></td>
<td>Shared objects and libraries</td>
</tr>
<tr>
<td><code>/opt/sqlanywhere11/res</code></td>
<td>String files</td>
</tr>
</tbody>
</table>

On AIX, the default root directory is `/usr/lpp/sqlanywhere11` instead of `/opt/sqlanywhere11`.

On Mac OS X, the default root directory is `/Applications/SQLAnywhere11/System` instead of `/opt/sqlanywhere11`.

- **File suffixes**  In the tables in this chapter, 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, `libdblib11.so` is a symbolic link to the file `libdblib11.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 libdblib11.so.1, there is a file named libdblib11\_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:

- libdbicu11.so.1
- libdbicu11\_r.so.1
- libdbicudt11.so.1
- sqlany.cvf

**Environment variables**  On Linux and Unix, 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/sqlanywhere11/bin32 and /opt/sqlanywhere11/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 SQLANY11.

For a description of 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 dbeng11.exe is a version 11 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, db1gen11.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.
<table>
<thead>
<tr>
<th>File extension</th>
<th>Platform</th>
<th>File type</th>
</tr>
</thead>
<tbody>
<tr>
<td>.bat, .cmd</td>
<td>Windows</td>
<td>Batch command files</td>
</tr>
<tr>
<td>.chm, .chw</td>
<td>Windows</td>
<td>Help system file</td>
</tr>
<tr>
<td>.dll</td>
<td>Windows</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>.exe</td>
<td>Windows</td>
<td>Executable file</td>
</tr>
<tr>
<td>.ini</td>
<td>All</td>
<td>Initialization file</td>
</tr>
<tr>
<td>.lic</td>
<td>All</td>
<td>License file</td>
</tr>
<tr>
<td>.lib</td>
<td>Varies by development tool</td>
<td>Static runtime libraries for the creation of embedded SQL executables</td>
</tr>
<tr>
<td>.res</td>
<td>Linux, Unix, Mac OS X</td>
<td>Language resource file for non-Windows environments</td>
</tr>
<tr>
<td>.so</td>
<td>Linux, Unix</td>
<td>Shared object or shared library file. The equivalent of a Windows DLL</td>
</tr>
<tr>
<td>.bundle, .dylib</td>
<td>Mac OS X</td>
<td>Shared object file. The equivalent of a Windows DLL</td>
</tr>
</tbody>
</table>

### 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 SQL Anywhere Deployment Wizard is the preferred tool for creating 32-bit 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

The Deployment Wizard does not include support for creating deployments of the 64-bit software components.

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 storage 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 also 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 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.

**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.
To create a deployment file

1. Start the Deployment Wizard:
   - From the Start menu, choose Programs » SQL Anywhere 11 » Deploy SQL Anywhere For Windows.
     or
   - From the Deployment subdirectory of your SQL Anywhere installation, run DeploymentWizard.exe.

2. Follow the instructions in the wizard.

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.

In Select Features, the available categories are Databases, Synchronization, and Administration Tools.

Databases

Use to select or deselect all of its subcategories.

- SQL Anywhere (32-bit) Use to select or deselect all of its subcategories.

  The following subcategories are available:

  o Client Interface Use to select or deselect all of its subcategories.
    - ODBC The SQL Anywhere ODBC driver.
    - Embedded SQL The SQL Anywhere Embedded SQL library.
    - OLEDB The SQL Anywhere OLE DB provider.
    - ADO.NET The SQL Anywhere .NET provider.
    - JDBC The SQL Anywhere JDBC driver.
    - Client Tools The SQL Anywhere client libraries like dblib11, dbtool11 and client utilities like dblocate, dbping, dbisqlc, and dbdsn.
    - Client Resources The SQL Anywhere language resource files like dblgen11, dblgde11, and dblges11 and the dblang language selection tool.

  o SQL Anywhere Server Use to select or deselect all of its subcategories.
    - Personal Server The SQL Anywhere personal server and license file.
    - Network Server The SQL Anywhere network server and license file.
    - Server Tools The SQL Anywhere server utilities like dbbackup, dberase, dbinit, dblog, dbsvc, dbunload and so on.
    - Unload Support Support for unloading version 9 and earlier databases.

- UltraLite Use to select or deselect all of its subcategories.

  The following subcategory is available:
UltraLite Engine
The UltraLite engine, utilities, and libraries such as uleng11, ulcond11, ulcreate, ulerase, ullgen11, ullgde11, ulrt11, and ulunload.

Synchronization
Use to select or deselect all of its subcategories.

- MobiLink Use to select or deselect all of its subcategories.
  - MobiLink Client The MobiLink Client tools and libraries such as dblnsn, dbmlsync, mlasinst, dbmlsynccli11, and the MobiLink .NET client provider.
  - MobiLink Server The MobiLink server, tools and libraries such as the server, ODBC driver, JDBC driver, and the MobiLink .NET provider.
- QAnywhere The QAnywhere application-to-application messaging tools.
- SQL Remote The SQL Remote tools and libraries including dbremote, dbxtract, and the message transport libraries such as dbsmtp11.

Administration Tools
Use to select or deselect all of its subcategories.

- Sybase Central The Sybase Central database manager and plugins. Use to select or deselect all of its subcategories.
  - SQL Anywhere Plugin The SQL Anywhere plugin.
  - MobiLink Plugin The MobiLink plugin.
  - UltraLite Plugin The UltraLite plugin.
  - QAnywhere Plugin The QAnywhere plugin.
- ISQL The Interactive SQL tool.
- DBConsole The administration and monitoring tool for database server connections.

If you would like to determine what files are included in each selectable component, create an MSI installer image selecting all components. A log file is created that details what files are included in every component. This text file can be examined with a text editor. You will see headings like "Feature: SERVER32_TOOLS" and "Feature: CLIENT32_TOOLS". The headings closely correspond to the Deployment Wizard components. This will give you an idea of what is included in each group.

To install a deployment file

- Use the Microsoft Windows Installer to install the deployment file. Here is a sample command:

  msiexec /package sqlany11.msi

A silent install can be performed using a command like the following:

  msiexec /qn /package sqlany11.msi SQLANYDIR=c:\sal1
• **/package <package-name>**  This parameter tells the Microsoft Windows Installer to install the specified package (in this case, *sqlany11.msi*).

• **/qn**  This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.

• **SQLANYDIR**  The value of this parameter is the path to where the software is to be installed.

### To uninstall a deployment

• It is also possible to perform a silent uninstall. The following is an example of a command line that would do this.

  ```
  msiexec /uninstall sqlany11.msi
  ```

  Alternately, a product code can be specified.

  ```
  msiexec.exe /qn /uninstall {19972A31-72EF-126F-31C7-5CF249B8593F}
  ```

• **/qn**  This parameter tells the Microsoft Windows Installer to operate in the background with no user interaction.

• **/uninstall <package-name> | <product-code>**  This parameter tells the Microsoft Windows Installer to uninstall the product associated with the specified MSI file or product code.

For more tips on how to do silent installs, see “Using a silent install for deployment” on page 1027.
Using a silent install for deployment

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 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 dialogue. 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=PEPEV-E96QE-A4000-00000-00000 INSTALLDIR=c:\sa11 DIR_SAMPLES=c:\sa11\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 options may be specified on the command line.

- **REGKEY**   The value of this parameter must be a valid software installation key.

- **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.

- **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").

The following example includes all the options:

```
d:\software\sqlanywhere\setup.exe /l:1033 /s "/v:/qn REGKEY=PEPEV-E96QE-A4000-00000-00000 INSTALLDIR=c:\sa11 DIR_SAMPLES=c:\sa11\Samples USERNAME="John Smith" COMPANYNAME="Smith Holdings"
```

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.

You can also install the documentation using a silent install. The *setup.exe* for this is in *d:\software Documentation*. An example of a command line that installs the documentation is:
d:\software\documentation\setup.exe /l:1033 /s "/v:/qn"

To generate an MSI log add the following to the command line after the /v:

/l*v! logfile

In the above 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. See the MSI documentation for details on how to reduce the output to the log file (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 {ECE263B0-6C8B-404C-B4AC-8FAB1C87AB4A}

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:

- {ECE263B0-6C8B-404C-B4AC-8FAB1C87AB4A} SQL Anywhere software
- {10964A7D-722B-4FE5-A16D-4977DCECEE95} SQL Anywhere documentation

The silent install described above does not address how you would select a subset of the components to install. This topic is better addressed by the Deployment Wizard. For information on component selection, see “Using the Deployment Wizard” on page 1023.
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 “Using the Deployment Wizard” on page 1023.

Deploying .NET clients

The simplest way to deploy .NET assemblies is to use the **Deployment Wizard**. For more information, see “Using the Deployment Wizard” on page 1023.

If your end users will be developing .NET applications, then consider integrating the SQL Anywhere .NET tools into Microsoft Visual Studio. On the client computer, you must do the following.

- Ensure Visual Studio is not running.
- Run `install-dir\Assembly\v2\SetupVSPackage.exe /install`.

If you want to create your own installation, this section describes the files to deploy to the end users.

Each .NET client computer must have the following:

- **A working .NET 2.0 (or later) installation**   Microsoft .NET assemblies and instructions for their redistribution are available from Microsoft Corporation. They are not described in detail here.
- **The SQL Anywhere .NET Data Provider**   The following table shows the files needed for a working SQL Anywhere .NET data provider. These files should be placed in a single directory.

The SQL Anywhere installation places the Windows assembly for the .NET Framework in the `Assembly\v2` subdirectory of your SQL Anywhere installation directory. The other files are placed in the operating-system subdirectory of your SQL Anywhere installation directory (for example, `bin32` or `bin64`).

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 subdirectory of your SQL Anywhere installation directory (for example, `ce\arm.50`).
Deploying databases and applications

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
<th>Windows Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET driver file</td>
<td>iAnywhere.Data.SQLAnywhere.dll</td>
<td>iAnywhere.Data.SQLAnywhere.dll</td>
</tr>
<tr>
<td>.NET Global Assembly Cache</td>
<td>N/A</td>
<td>iAnywhere.Data.SQLAnywhere.gac</td>
</tr>
<tr>
<td>Connect window</td>
<td>dbcon11.dll</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The table above shows files with the designation [xx]. 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 about deploying the SQL Anywhere .NET provider, see “Deploying the SQL Anywhere .NET Data Provider” on page 139.

### Deploying OLE DB and ADO clients

The simplest way to deploy OLE DB client libraries is to use the Deployment Wizard. For more information, see “Using the Deployment Wizard” on page 1023.

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 (dboledba11) is an assist DLL used to provide schema support.

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE DB driver file</td>
<td>dboledb11.dll</td>
</tr>
<tr>
<td>OLE DB driver file</td>
<td>dboledba11.dll</td>
</tr>
<tr>
<td>Language-resource library</td>
<td>dbg[xx]11.dll</td>
</tr>
<tr>
<td>Connect window</td>
<td>dbcon11.dll</td>
</tr>
<tr>
<td>Elevated operations agent</td>
<td>dbelevate11.exe (Vista only)</td>
</tr>
</tbody>
</table>
The table above shows a file with the designation [xx]. 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.

OLE DB providers require many registry entries. You can make these by self-registering the dboledb11.dll and dboledba11.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 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:

```vbnet
connectString = "Provider=SAOLEDB;DSN=SQL Anywhere 11 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.

```c++
hr = db.Open(_T("SAOLEDB.11"), &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]
@="{41dfe9f3-db91-11d2-8c43-006008d26a6f}"

[HKEY_CLASSES_ROOT\SAOLEDB\CurVer]
@="SAOLEDB.11"
```

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:\sa11\bin64\dboledb11.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.

In the following exercise, you generate a unique set of GUIDs, choose a unique provider name and unique DLL names. These three things will help you create a unique OLE DB provider which you can deploy with your application.

Here are the steps involved in creating a custom version of the OLE DB provider.

**To customize the OLE DB provider**

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID1</td>
<td>41dfe9f3-db92-11d2-8c43-006008d26a6f</td>
</tr>
<tr>
<td>GUID2</td>
<td>41dfe9f4-db92-11d2-8c43-006008d26a6f</td>
</tr>
<tr>
<td>GUID3</td>
<td>41dfe9f5-db92-11d2-8c43-006008d26a6f</td>
</tr>
<tr>
<td>GUID4</td>
<td>41dfe9f6-db92-11d2-8c43-006008d26a6f</td>
</tr>
</tbody>
</table>

   Note that it is the first part of the GUID (for example, 41dfe9f3) that is incrementing.
4. 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).

5. 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.

<table>
<thead>
<tr>
<th>Version 10 or later</th>
<th>Version 9 or earlier</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAOLEDB</td>
<td>ASAProv</td>
</tr>
<tr>
<td>SAErrorLookup</td>
<td>ASAErrorLookup</td>
</tr>
<tr>
<td>SAEnum</td>
<td>ASAEnum</td>
</tr>
<tr>
<td>SAOLEDBA</td>
<td>ASAProvA</td>
</tr>
</tbody>
</table>

6. 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).

<table>
<thead>
<tr>
<th>SQL Anywhere provider</th>
<th>Your custom provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAOLEDB</td>
<td>Acme</td>
</tr>
<tr>
<td>SAErrorLookup</td>
<td>AcmeErrorLookup</td>
</tr>
<tr>
<td>SAEnum</td>
<td>AcmeEnum</td>
</tr>
<tr>
<td>SAOLEDBA</td>
<td>AcmeA</td>
</tr>
</tbody>
</table>

7. Make copies of the SQL Anywhere provider DLLs (dboledb11.dll and dboledba11.dll) under different names. Note that there is no dboledba11.dll for Windows Mobile.

   copy dboledb11.dll myoledb11.dll
   copy dboledba11.dll myoledba11.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 dboledb11.dll or dboledba11.dll). If you name the provider DLL myoledb11 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 myoledba11 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.
<table>
<thead>
<tr>
<th>DLL name(s) chosen</th>
<th>Corresponding HKEY_CLASSES_ROOT\name</th>
</tr>
</thead>
<tbody>
<tr>
<td>myoledb11.dll</td>
<td>HKEY_CLASSES_ROOT\myoledb11</td>
</tr>
<tr>
<td>myoledba11.dll</td>
<td>HKEY_CLASSES_ROOT\myoledba11</td>
</tr>
<tr>
<td>acmeOledb.dll</td>
<td>HKEY_CLASSES_ROOT\acmeOledb</td>
</tr>
<tr>
<td>acmeOledba.dll</td>
<td>HKEY_CLASSES_ROOT\acmeOledba</td>
</tr>
<tr>
<td>SAcustom.dll</td>
<td>HKEY_CLASSES_ROOT\SAcustom</td>
</tr>
<tr>
<td>SAcustomA.dll</td>
<td>HKEY_CLASSES_ROOT\SAcustomA</td>
</tr>
</tbody>
</table>

8. Use the search/replace capability of an editor to change all the occurrences of `myoledb11` and `myoledba11` 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:\mypath\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 will have to 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\myoledb11]
@="Custom SQL Anywhere OLE DB Provider 11.0"

[HKEY_CLASSES_ROOT\myoledb11\Clsid]
@="{GUID1}"

; Data1 of the following GUID must be 3 greater than the previous, for example, 41dfe9f3 + 3 => 41dfe9ee.

[HKEY_CLASSES_ROOT\myoledba11]
@="Custom SQL Anywhere OLE DB Provider 11.0"

[HKEY_CLASSES_ROOT\myoledba11\Clsid]
@="{GUID4}"

; Current version (or version independent prog ID) entries (what you get when you have "SQLAny"
; instead of "SQLAny.11")

[HKEY_CLASSES_ROOT\SQLAny]
@="SQL Anywhere OLE DB Provider"

[HKEY_CLASSES_ROOT\SQLAny\Clsid]
@="{GUID1}"

[HKEY_CLASSES_ROOT\SQLAny\CurVer]
@="SQLAny.11"
```
[HKEY_CLASSES_ROOT\SQLAnyEnum]  
@="SQL Anywhere OLE DB Provider Enumerator"

[HKEY_CLASSES_ROOT\SQLAnyEnum\Clsid]  
@="{GUID2}" 

[HKEY_CLASSES_ROOT\SQLAnyEnum\CurVer]  
@="SQLAnyEnum.11"

[HKEY_CLASSES_ROOT\SQLAnyErrorLookup]  
@="SQL Anywhere OLE DB Provider Extended Error Support"

[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\Clsid]  
@="{GUID3}" 

[HKEY_CLASSES_ROOT\SQLAnyErrorLookup\CurVer]  
@="SQLAnyErrorLookup.11"

[HKEY_CLASSES_ROOT\SQLAnyA]  
@="SQL Anywhere OLE DB Provider Assist"

[HKEY_CLASSES_ROOT\SQLAnyA\Clsid]  
@="{GUID4}" 

[HKEY_CLASSES_ROOT\SQLAnyA\CurVer]  
@="SQLAnyA.11"

; Standard entries (Provider=SQLAny.11)

[HKEY_CLASSES_ROOT\SQLAny.11]  
@="Sybase SQL Anywhere OLE DB Provider 11.0"

[HKEY_CLASSES_ROOT\SQLAny.11\Clsid]  
@="{GUID1}" 

[HKEY_CLASSES_ROOT\SQLAnyEnum.11]  
@="Sybase SQL Anywhere OLE DB Provider Enumerator 11.0"

[HKEY_CLASSES_ROOT\SQLAnyEnum.11\Clsid]  
@="{GUID2}" 

[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.11]  
@="Sybase SQL Anywhere OLE DB Provider Extended Error Support 11.0"

[HKEY_CLASSES_ROOT\SQLAnyErrorLookup.11\Clsid]  
@="{GUID3}" 

[HKEY_CLASSES_ROOT\SQLAnyA.11]  
@="Sybase SQL Anywhere OLE DB Provider Assist 11.0"

[HKEY_CLASSES_ROOT\SQLAnyA.11\Clsid]  
@="{GUID4}" 

; SQLAny (Provider=SQLAny.11)

[HKEY_CLASSES_ROOT\CLSID\{GUID1}]  
@="SQLAny.11"
"OLEDB_SERVICES"=dword:ffffffff

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors]  
@="Extended Error Service"

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ExtendedErrors\{GUID3}]
Deploying databases and applications

@="Sybase SQL Anywhere OLE DB Provider Error Lookup"

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\InprocServer32]
@="d:\mypath\bin32\myoledb11.dll"
"ThreadingModel"="Both"

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\OLE DB Provider]
@="Sybase SQL Anywhere OLE DB Provider 11.0"

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\ProgID]
@="SQLAny.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID1}\VersionIndependentProgID]
@="SQLAny"

; SQLAnyErrorLookup

[HKEY_CLASSES_ROOT\CLSID\{GUID3}\{GUID1}\OLE DB Provider Error Lookup 11.0"
@="SQLAnyErrorLookup.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID3}\InprocServer32]
@="d:\mypath\bin32\myoledb11.dll"
"ThreadingModel"="Both"

[HKEY_CLASSES_ROOT\CLSID\{GUID3}\ProgID]
@="SQLAnyErrorLookup.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID3\}\VersionIndependentProgID]
@="SQLAnyErrorLookup"

; SQLAnyEnum

[HKEY_CLASSES_ROOT\CLSID\{GUID2\}]
@="SQLAnyEnum.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID2\}\InprocServer32]
@="d:\mypath\bin32\myoledb11.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.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID2\}\VersionIndependentProgID]
@="SQLAnyEnum"

; SQLAnyA

[HKEY_CLASSES_ROOT\CLSID\{GUID4\}]
@="SQLAnyA.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID4\}\InprocServer32]
@="d:\mypath\bin32\myoledba11.dll"
"ThreadingModel"="Both"

[HKEY_CLASSES_ROOT\CLSID\{GUID4\}\ProgID]
@="SQLAnyA.11"

[HKEY_CLASSES_ROOT\CLSID\{GUID4\}\VersionIndependentProgID]
@="SQLAnyA"
Deploying ODBC clients

The simplest way to deploy ODBC clients is to use the Deployment Wizard. For more information, see “Using the Deployment Wizard” on page 1023.

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.

- **The SQL Anywhere ODBC driver**    The files that must be included in a deployment of an ODBC client application are described next in ODBC driver required files.

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".

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>dbodbc11.dll</td>
</tr>
<tr>
<td></td>
<td>dbcon11.dll</td>
</tr>
<tr>
<td></td>
<td>dbicu11.dll</td>
</tr>
<tr>
<td></td>
<td>dbicudt11.dll</td>
</tr>
<tr>
<td></td>
<td>dblg[xx]11.dll</td>
</tr>
<tr>
<td></td>
<td>dbelevate11.exe</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>dbodbc11.dll</td>
</tr>
<tr>
<td></td>
<td>dbicu11.dll (optional)</td>
</tr>
<tr>
<td></td>
<td>dbicudt11.dat (optional)</td>
</tr>
<tr>
<td></td>
<td>dblg[xx]11.dll</td>
</tr>
<tr>
<td>Platform</td>
<td>Required files</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Linux, Solaris, HP-UX</td>
<td><code>libdbodbc11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_n.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>dblg[xx]11.res</code></td>
</tr>
<tr>
<td>Linux, Solaris, HP-UX MT</td>
<td><code>libdbodbc11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_r.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11_r.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11_r.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.so.1</code></td>
</tr>
<tr>
<td></td>
<td><code>dblg[xx]11.res</code></td>
</tr>
<tr>
<td>AIX</td>
<td><code>libdbodbc11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_n.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>dblg[xx]11.res</code></td>
</tr>
<tr>
<td>AIX MT</td>
<td><code>libdbodbc11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_r.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11_r.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11_r.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.so</code></td>
</tr>
<tr>
<td></td>
<td><code>dblg[xx]11.res</code></td>
</tr>
<tr>
<td>Platform</td>
<td>Required files</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mac OS X</td>
<td><code>dbodbc11.bundle</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_n.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>dibl[xx]11.res</code></td>
</tr>
<tr>
<td>Mac OS X MT</td>
<td><code>dbodbc11_r.bundle</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodbc11_r.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbodm11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbtasks11_r.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicu11_r.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>libdbicudt11.dylib</code></td>
</tr>
<tr>
<td></td>
<td><code>dibl[xx]11.res</code></td>
</tr>
</tbody>
</table>

**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 `libdbodm11`.
- Note that for Windows Vista or later versions of Windows, you must include the SQL Anywhere elevated operations agent (`dbelevate11.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 `[xx]`. 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 Windows, the Connect window support code (`dbcon11.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.
Configuring the ODBC driver

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. To register the 32-bit and 64-bit versions of the ODBC driver, open up a command prompt and issue the following commands.

```
regsvr32 install-dir\bin32\dbodbc11.dll
regsvr32 install-dir\bin64\dbodbc11.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 11
```

Sample values for 32-bit Windows are shown below:

<table>
<thead>
<tr>
<th>Value name</th>
<th>Value type</th>
<th>Value data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>String</td>
<td>install-dir\bin32\dbodbc11.dll</td>
</tr>
<tr>
<td>Setup</td>
<td>String</td>
<td>install-dir\bin32\dbodbc11.dll</td>
</tr>
</tbody>
</table>

There is also a registry value in the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers
```

The value is as follows:

<table>
<thead>
<tr>
<th>Value name</th>
<th>Value type</th>
<th>Value data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Anywhere 11</td>
<td>String</td>
<td>Installed</td>
</tr>
</tbody>
</table>
64-bit Windows

For 64-bit Windows, the 32-bit ODBC driver registry entries ("SQL Anywhere 11" and "ODBC Drivers") will be 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.

Deploying connection information

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 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.

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.

User data source definitions are stored in the part of the registry containing settings for the specific user currently logged on to the system. 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 connection by providing DSN=MyApp in the ODBC connection string.

File data sources are not held in the registry, but are held in a special directory. A connection string must provide a FileDSN connection parameter to use a File data source.
Data source registry entries (Windows)

Each user data source is identified to the system by registry entries. 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 key is as follows:

```
HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI\user-data-source-name
```

For System data sources, the 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. For example, the SQL Anywhere 11 Demo key corresponding to the SQL Anywhere 11 Demo system Data Source Name (DSN) contains the following settings for 32-bit Windows:

<table>
<thead>
<tr>
<th>Value name</th>
<th>Value type</th>
<th>Value data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autostop</td>
<td>String</td>
<td>YES</td>
</tr>
<tr>
<td>DatabaseFile</td>
<td>String</td>
<td><code>samples-dir\demo.db</code></td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>SQL Anywhere 11 Sample Database</td>
</tr>
<tr>
<td>Driver</td>
<td>String</td>
<td><code>install-dir\bin32\dbodbc11.dll</code></td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>sql</td>
</tr>
<tr>
<td>ServerName</td>
<td>String</td>
<td>demo11</td>
</tr>
<tr>
<td>StartLine</td>
<td>String</td>
<td><code>install-dir\bin32\dbeng11.exe</code></td>
</tr>
<tr>
<td>UserID</td>
<td>String</td>
<td>DBA</td>
</tr>
</tbody>
</table>

**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 in the event that a computer is running multiple SQL Anywhere database servers and can help prevent timing-dependent connection failures.
In these entries, *install-dir* is the SQL Anywhere installation directory. For 64-bit Windows, *bin32* would be replaced by *bin64*.

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 11 Demo, and has the following value:

<table>
<thead>
<tr>
<th>Value name</th>
<th>Value type</th>
<th>Value data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Anywhere 11 Demo</td>
<td>String</td>
<td>SQL Anywhere 11</td>
</tr>
</tbody>
</table>

**Caution: ODBC 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 Current 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 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 DSN.
Contains only the name and location of the ODBC driver. | The name of the database file (DBF=) and/or the database server (ENG=). Optionally, it may contain other connection parameters such as Userid (UID=) and PASSWORD (PWD=). | User ID and password if not provided in the DSN 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 (ENG=). 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.

For more information about ODBC connections and configurations, see the following:

- “SQL Anywhere database connections” [SQL Anywhere Server - Database Administration].
- The Open Database Connectivity (ODBC) SDK, available from Microsoft.

**Deploying embedded SQL clients**

The simplest way to deploy embedded SQL clients is to use the **Deployment Wizard**. For more information, see “Using the Deployment Wizard” on page 1023.

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.
Installing files for embedded SQL clients

The following table shows which files are needed for embedded SQL clients.

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface library</td>
<td>dblib11.dll</td>
<td>libdblib11_r.so</td>
<td>libdblib11_r.dylib</td>
</tr>
<tr>
<td>Thread support library</td>
<td>N/A</td>
<td>libdbtasks11_r.so</td>
<td>libdbtasks11_r.dylib</td>
</tr>
<tr>
<td>Connect window</td>
<td>dbcon11.dll</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Notes

- The table above shows files with the designation [xx]. 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 non-multi-threaded applications on Linux/Unix, use libdblib11.so and libdbtasks11.so.

- For non-multi-threaded applications on Mac OS X, use libdblib11.dylib and libdbtasks11.dylib.

- If the client application uses encryption then the appropriate encryption support (dbec11.dll, dbfips11.dll, or dbrsa11.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 more information about deploying ODBC information, see “Deploying ODBC clients” on page 1037.

- The Connect window support (dbcon11.dll) is needed if your end users will be creating their own data sources, if they will 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.

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.
Deploying JDBC clients

You must install a Java Runtime Environment to use JDBC. Version 1.6.0 or later is recommended.

In addition to a Java Runtime Environment, each JDBC client requires the iAnywhere JDBC driver or jConnect.

iAnywhere JDBC driver

To deploy the iAnywhere JDBC driver, you must deploy the following files:

- **jodbc.jar** This must be in the application's classpath. This file is located in the SQL Anywhere installation *java* folder.

- **dbjodbc11.dll** This must be locatable in the system path. On Linux and Unix environments, the file is a shared library called *libdbjodbc11.so*. On Mac OS X, the file is a shared library called *libdbjodbc11.dylib*.

- The ODBC driver files. For more information, see “ODBC driver required files” on page 1037.

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 jConnect for JDBC.

- 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 *dbrsakp11.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 *libdbrsakp11.so*. On Mac OS X, the file is a shared library called *libdbrsakp11.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.

For more information about URLs, see “Supplying a URL to the driver” on page 509.

Deploying Open Client applications

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 dbrsakp11.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 libdbrsakp11.so. On Mac OS X, the file is a shared library
called libdbrsakp11_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 “Configuring Open Servers” [SQL Anywhere Server -
Database Administration].
Deploying administration tools

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 more information, see “Using the Deployment Wizard” on page 1023.

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 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 (for example, scjview -batch) and examining the file that is created. 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 (for example, scjview -batch) and examining the file that is created. The APPLICATION_ARGUMENTS specification is optional.

Here are the contents of a sample initialization file for Sybase Central.

```
JRE_DIRECTORY=c:\Sun\JRE160_x86
VM_ARGUMENTS=-Xmx200m
JAR_PATHS=c:\scj\jars;c:\scj\jhelp
ADDITIONAL_CLASSPATH=
```
This scenario assumes that a copy of the 32-bit Sun JRE is located in `c:\Sun\JRE160_x86`. As well, the Sybase Central executable and shared libraries (DLLs) like `jsyblib600` are stored in `c:\scj\bin`. The SQL Anywhere JAR files are stored in `c:\scj\jars`. The Sun JavaHelp 2.0 JAR files are stored in `c:\scj\help`.

**Note**

When you are deploying applications, the personal database server (dbeng11) 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.

### Deploying administration tools on Windows without InstallShield

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 InstallShield. 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 11.0.1 and cannot be applied to earlier or later versions of the software.

**Check your license agreement**

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.

**Modifying your registry is dangerous**

Modify your registry 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 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

**Note**
To check your JRE version on Mac OS X, go to the Apple menu, and then choose 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 developer.apple.com/java/download/.

The instructions in the following sections are structured so that you can install any (or all) of these six 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:\sa11` is used as the example installation folder. The software must be installed into a directory tree structure having the following layout:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sa11</code></td>
<td>The root folder. While the following steps assume you are installing into <code>c:\sa11</code>, you are free to put the directory anywhere (for example, <code>C:\Program Files \SQLAny11</code>).</td>
</tr>
</tbody>
</table>
### Directory Description

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sa1 java</td>
<td>Holds Java program JAR files.</td>
</tr>
<tr>
<td>sa1 bin32</td>
<td>Holds the native 32-bit Windows components used by the program, including the programs that launch the applications.</td>
</tr>
<tr>
<td>sa1 Sun\JavaHelp-2_0</td>
<td>The JavaHelp runtime library.</td>
</tr>
<tr>
<td>sa1 Sun\jre160_x86</td>
<td>The 32-bit Java Runtime Environment.</td>
</tr>
</tbody>
</table>

### x64

On most platforms, the Java-based administration tools are 32-bit applications. Except for Mac OS X, there are no 64-bit versions. The 32-bit administration tools can be deployed on x64-based platforms where the 32-bit JRE is available.

### Itanium 64

There are no Java-based administration tools available for deployment on the Itanium (ia64) platform. However, there is a native version of Interactive SQL that is not as feature-rich as the Java version. See “Deploying dbisqlc” on page 1072.

The following table lists the files required for each of the software bundles. Make a list of the files you need, and then copy them into the directory structure outlined above. In general, you should take the files from an already-installed copy of SQL Anywhere.

### Table of Files

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentation[xx]htmlhelp\sqlanywhere_[xx]11.chm</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>documentation\sqlanywhere_[xx]11.map</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>c:\windows\system32\keyHH.exe1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\jodbc.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\JComponents1101.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\jlogon.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>File</td>
<td>Interactive SQL</td>
<td>Sybase Central with the SQL Anywhere plug-in</td>
<td>Sybase Central with the Mobi-Link plug-in</td>
<td>Sybase Central with the QAnywhere plug-in</td>
<td>Sybase Central with the UltraLite plug-in</td>
<td>SQL Anywhere Console</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------</td>
<td>------------------------------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>javaSCEditor600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>javajsyblib600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SunJavaHelp-2_0jh.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sun\jre160_x86..</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\jsyblib600.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dlib11.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbsd3bc11.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbsdcon11.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbflg[xx]11.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbtool11.dll</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbisql.exe (Vista and later)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\dbisql.com</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbisql.exe</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\isql.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\saip11.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\scjview.exe</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin32\scvw[xx]600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\sybasecentral600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\sailib.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java\sapplugin.jar</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Deploying administration tools

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>java\debugger.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbput11.dll</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\apache_files.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\apache_license_1.1.txt</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\apache_license_2.0.txt</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\log4j.jar</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\mlplugin.jar</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\mldesign.jar</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\stax-api-1.0.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\wstx-asl-3.2.6.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\velocity.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\velocity-dep.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\qaplugin.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\qaconnector.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\mlstream.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\qaagent.exe</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbicu11.dll</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbicud11.dll</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbghelp.dll</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32\dbinit.exe</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java\ulplugin.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
The exact name of the Windows system directory differs depending on which operating system you are using.

The table above shows files with the designation [xx]. 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 the following section International message and context-sensitive help files.

Some file paths above end with "...". This indicates that the entire tree, including subdirectories, should be copied.

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. Copy the 32-bit version of the JRE files from the `install-dir\Sun\jre160_x86` directory. Copy the entire `jre160_x86` tree, including subdirectories.

For reference, the `sqlanywhere.jpr` file contains a list of the jar files for the SQL Anywhere plug-in of Sybase Central.

The `mobilink.jpr` file contains a list of the jar files for the MobiLink plug-in of Sybase Central.

The `qanywhere.jpr` file contains a list of the jar files for the QAnywhere plug-in of Sybase Central. When deploying the QAnywhere plug-in, `dbinit` is required. For information about deploying database tools, see “Deploying database utilities” on page 1082.

The `ultralite.jpr` file contains a list of the jar files for the UltraLite plug-in of Sybase Central.

### 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. The translated files are as follows:

<table>
<thead>
<tr>
<th>File</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblgen11.dll</td>
<td>English</td>
</tr>
<tr>
<td>dblgde11.dll</td>
<td>German</td>
</tr>
</tbody>
</table>
You must also add the context-sensitive help files for those other languages. The available translated files are as follows:

<table>
<thead>
<tr>
<th>File</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>scvwen600.jar</td>
<td>English</td>
</tr>
<tr>
<td>scvwde600.jar</td>
<td>German</td>
</tr>
<tr>
<td>scvwfr600.jar</td>
<td>French</td>
</tr>
<tr>
<td>scvwja600.jar</td>
<td>Japanese</td>
</tr>
<tr>
<td>scvwzh600.jar</td>
<td>Simplified Chinese</td>
</tr>
</tbody>
</table>

These files are included with localized versions of SQL Anywhere.

**Step 3: Registering the administration tools with Windows**

You must set the following registry values for the administration tools. The names of the registry values are case sensitive.

- In `HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\Sybase Central\6.0.0`
  - **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\11.0`
  - **Location** The fully-qualified path to the root of the installation folder (`C:\Program Files\SQL Anywhere 11` by default) containing the Sybase Central files.
  - **Language** The two-letter code for the language used by SQL Anywhere. 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, these registry entries are in the 32-bit registry (`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 our example installation folder of `c:\sa11`, the following is a sample `.reg` file:

```
REGEDIT4
```
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 the `c:\sa11\bin32` directory 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]
```

When you deploy Interactive SQL or Sybase Central, add the following directory to the existing path:

```
c:\sa11\bin32
```

**Step 5: Registering the Sybase Central plug-ins**

This step involves the configuration of Sybase Central. If you are not installing Sybase Central, you can skip it.

Sybase Central requires a configuration file that lists the installed plug-ins. Your installer must create this file. Note that it contains full paths to several JAR files that may change depending on where the software is installed.

The file is called `.scRepository600`. On Windows XP/200x, it resides in the `%allusersprofile%\application data\Sybase Central 6.0.0` folder. On Windows Vista, it resides in the `%ProgramData%\Sybase Central 6.0.0` folder. It is a plain text file that contains some basic information about the plug-ins that Sybase Central should load.

On Windows Vista, all users should have read access to the directory that contains the `.scRepository600` file. This can be done using the following command. To do this manually, open an administrator command prompt window (right-click Command Prompt and click Run As Administrator).

```
icacls "%ProgramData%\Sybase Central 6.0.0" /grant everyone:F
```

The provider information for SQL Anywhere is created in the repository file using the following commands.

```
scjview.exe -register "C:\Program Files\SQL Anywhere 11\java\sqlanywhere.jpr"
```

The contents of the `sqlanywhere.jpr` file looks like this (some entries have been split across multiple lines for display purposes). The **AdditionalClasspath** lines must be entered on a single line in the .jpr file.
The \textit{sqlanywhere.jpr} file was created in the \textit{java} folder of the SQL Anywhere installation when you originally installed SQL Anywhere. Use it as the model for the \textit{jpr} file that you must create as part of the install process. There are also versions of this file for MobiLink, QAnywhere, and UltraLite called \textit{mobilink.jpr}, \textit{qanywhere.jpr}, and \textit{ultralite.jpr} respectively. They are also located in the \textit{java} folder.

Here is a portion of the \textit{.scRepository600} 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: 6.0.0.1154
# Fri Feb 22 10:22:20 EST 2008
#
SCRRepositoryInfo/Version=4
#
Providers/sqlanywhere1100/Version=11.0.1.1297
Providers/sqlanywhere1100/UseClassloader=true
Providers/sqlanywhere1100/ClassLoaderId=SA1100
Providers/sqlanywhere1100/Classpath= C:\Program Files\SQL Anywhere 11\java\saplugin.jar
Providers/sqlanywhere1100/Name=SQL Anywhere 11
Providers/sqlanywhere1100/AdditionalClasspath= C:\Program Files\SQL Anywhere 11\java\isql.jar;
  C:\Program Files\SQL Anywhere 11\java\salib.jar;
  C:\Program Files\SQL Anywhere 11\java\JComponents1101.jar;
  C:\Program Files\SQL Anywhere 11\java\jlogon.jar;
  C:\Program Files\SQL Anywhere 11\java\debugger.jar;
  C:\Program Files\SQL Anywhere 11\java\jodbc.jar
```

\textbf{Notes}

- Your installer should create a file similar to this one using the techniques described above. The only changes required are the fully-qualified paths to the JAR files in the Classpath and AdditionalClasspath lines.
- The AdditionalClasspath lines shown above have wrapped to take up additional lines. They must be on a single line in the \textit{.scRepository600} file.
- Backslash characters (\) are represented with an escape sequence of \ in the \textit{.scRepository600} file.
- The first line indicates the version of the \textit{.scRepository600} file.
- The lines beginning with # are comments.
Step 6: 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 11 Demo** is created in the `.scRepository600` 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 11 Demo** connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Name" "SQL Anywhere 11 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/FirstTimeStart" "false"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Description" "Suitable Description"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/ProviderId" "sqlanywhere1100"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Provider" "SQL Anywhere 11"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileSettings" "DSN\eSQL\0020Anywhere\002011\0020Demo;UID \eDBA;PWD\e35c624d517fb"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileName" "SQL Anywhere 11 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the `.scRepository600` file. Define a connection profile using Sybase Central and then look at the `.scRepository600` file for the corresponding lines.

Here is a portion of the `.scRepository600` 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: 6.0.0.1154
#  Fri Feb 23 13:09:14 EST 2007
#  
ConnectionProfiles/SQL Anywhere 11 Demo/Name=SQL Anywhere 11 Demo
ConnectionProfiles/SQL Anywhere 11 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 11 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 11 Demo/ProviderId=sqlanywhere1100
ConnectionProfiles/SQL Anywhere 11 Demo/Provider=SQL Anywhere 11
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileSettings=DSN\eSQL\0020Anywhere\002011\0020Demo;UID \eDBA;PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileName=SQL Anywhere 11 Demo
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileType=SQL Anywhere
```
Step 7: Registering the SQL Anywhere ODBC driver

You must install the SQL Anywhere ODBC driver before it can be used by the iAnywhere JDBC Driver in the administration tools.

For more information, see “Configuring the ODBC driver” on page 1040.

Deploying administration tools 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 11.0.1 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 “Deploying dbisqlc” on page 1072.

Check your license agreement
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 you want 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
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.

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. All files must be installed below the `/opt/sqlanywhere11/` directory.

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 following table lists the files required for each of the software bundles on Linux and Sun Solaris. Make a list of the files you need, and then copy them into the directory structure outlined above. In general, you should take the files from an already-installed copy of SQL Anywhere.

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in [1]</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java/jdbc.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>java/JComponents1101.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>java/jlogon.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>java/SCEditor600.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>java/jsylib600.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>lib32/libjsylib600_r.so.1</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>sun/javahelp-2_0/fjh.jar</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>jre_1.6.0_linux_sun_i586/...</code> (Linux only)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
## Deploying administration tools

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in [1]</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>jre_1.6.0_solaris_sun_sparc/... (Solaris only)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdblib11_r.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbjodbc11.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbodbc11_r.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbodm11.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbtasks11_r.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>res/dblg[xx]11.res</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbtool11_r.so.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/dbisql</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/isql.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/scjview</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/scvw[xx]600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/sybasecentral600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/salib.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/sapplugin.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/debugger.jar</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>lib32/libdbput11_r.so.1</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>lib32/libmljodbc11.so.1</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/apache_files.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/apache_license_1.1.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/apache_license_2.0.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Deploying databases and applications

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in [1]</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>java/log4j.jar</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/mlplugin.jar</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/mldesign.jar</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/stax-api-1.0.jar</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/wstx-asl-3.2.6.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/velocity.jar</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/velocity-dep.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/qapplugin.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/qaconnector.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>java/mlstream.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib32/libdbicu11_r.so</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib32/libdbicudt11.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32/dbinit</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/ulplugin.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>lib32/libulscp11_r.so.1</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib32/libulhltool11_r.so.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>res/ulg[xx]11.res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/uleng11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulcreate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulcreate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulunload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulunload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
The table above shows files with the designation [xx]. For Linux only, 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 the following section “International message and context-sensitive help files” on page 1067.

The following table lists the files required for each of the software bundles on Mac OS X. Make a list of the files you need, and then copy them into the directory structure outlined above. In general, you should take the files from an already-installed copy of SQL Anywhere.

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>Sybase Central with the UltraLite plug-in [1]</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin32/ulsync</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulinit</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulvalid</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/ulerase</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>bin32/dbconsole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/DBConsole.jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

[1] UltraLite is only supported on Linux.
### Deploying databases and applications

<table>
<thead>
<tr>
<th>File</th>
<th>Interactive SQL</th>
<th>Sybase Central with the SQL Anywhere plug-in</th>
<th>Sybase Central with the MobiLink plug-in</th>
<th>Sybase Central with the QAnywhere plug-in</th>
<th>SQL Anywhere Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun/javahelp-2_0/jh.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdblib11_r.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbhjodbc11.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbodbc11_r.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbodm11.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbtasks11_r.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>res/dblgen11.res</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbtool11_r.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin64/dbisql</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/isql.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin64/scjview</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>bin64/scvwen600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/sybasecentral600.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/salib.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/sapplugin.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/debugger.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lib64/libdbput11_r.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>libmljodbc11.dylib</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/apache_files.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/apache_license_1.1.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/apache_license_2.0.txt</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>java/log4j.jar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>File</td>
<td>Interactive SQL</td>
<td>Sybase Central with the SQL Anywhere plug-in</td>
<td>Sybase Central with the MobiLink plug-in</td>
<td>Sybase Central with the QAnywhere plug-in</td>
<td>SQL Anywhere Console</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------</td>
<td>------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>java/mlplugin.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/mldesign.jar</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/stax-api-1.0.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/wstx-asl-3.2.6.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/velocity.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/velocity-dep.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/qapugin.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/qaconnector.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/mlstream.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib64/libdicu11_r.dylib</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib64/libdbicudt11.dylib</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin32/dbinit</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin64/dbconsole</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java/DBConsole.jar</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 11. Copy the entire tree, including subdirectories.

The `sqlanywhere.jpr` file contains a list of the jar files for the SQL Anywhere plug-in of Sybase Central. The `mobilink.jpr` file contains a list of the jar files for the MobiLink plug-in of Sybase Central.
The *qanywhere.jpr* file contains a list of the jar files for the QAnywhere plug-in of Sybase Central. When deploying the QAnywhere plug-in, dbinit is required. For information about deploying database tools, see “Deploying database utilities” on page 1082.

The *ultralite.jpr* file contains a list of the jar files for the UltraLite plug-in of Sybase Central.

The creation of several links is required for the bundles in the tables above. The following sections provide details.

**Mac OS X**

Note that on Mac OS X, shared objects have a `.dylib` extension. Symlink (symbolic link) creation is necessary for the following dylibs:

- `libdbjodbc11.jnilib -> libdbjodbc11.dylib`
- `libdblib11_r.jnilib -> libdblib11_r.dylib`
- `libdbput11_r.jnilib -> libdbput11_r.dylib`
- `libmljodbc11.jnilib -> libmljodbc11.dylib`

**Linux/Solaris Base component files**

All bundles require the links listed in this section.

Create the following symbolic links in `/opt/sqlanywhere11/lib32`:

- `libdbicu11_r.so -> libdbicu11_r.so.1`
- `libdbicudt11.so -> libdbicudt11.so.1`
- `libdbjodbc11.so -> libdbjodbc11.so.1`
- `libdbodbc11_r.so -> libdbodbc11_r.so.1`
- `libdbodm11.so -> libdbodm11.so.1`
- `libdbtasks11_r.so -> libdbtasks11_r.so.1`

Create a symbolic link in `/opt/sqlanywhere11/sun`. The symbolic link for Linux is `jre160_x86` for the 32-bit JRE. The symbolic link for other systems is `jre_160`.

- `jre160_x86 -> /opt/sqlanywhere11/sun/jre_1.6.0_linux_sun_i586` (Linux)
- `jre160 -> /opt/sqlanywhere11/sun/jre_1.6.0_solaris_sun_sparc` (Solaris)

**Linux/Solaris Interactive SQL files**

Create the following symbolic links in `/opt/sqlanywhere11/lib32`:

- `libdblib11_r.so -> libdblib11_r.so.1`
- `libdbtool11_r.so -> libdbtool11_r.so.1`

**Linux/Solaris Sybase Central with the SQL Anywhere plug-in**

Create the following symbolic links in `/opt/sqlanywhere11/lib32`:

- `libdblib11_r.so -> libdblib11_r.so.1`
- `libdbput11_r.so -> libdbput11_r.so.1`
- `libdbtool11_r.so -> libdbtool11_r.so.1`

**Linux/Solaris Sybase Central with the MobiLink plug-in**

Create the following symbolic links in `/opt/sqlanywhere11/lib32`:
libdblib11_r.so -> libdblib11_r.so.1
libdbmlput11_r.so -> libdbmlput11_r.so.1
libdbtool11_r.so -> libdbtool11_r.so.1

Create an additional symbolic link in /opt/sqlanywhere11/sun for 64-bit Linux. The symbolic link for Linux is jre160_x64 for the 64-bit JRE.

jre160_x64 -> /opt/sqlanywhere11/sun/jre_1.6.0_linux_sun_x64 (Linux)

Linux/Solaris Sybase Central with the QAnywhere plug-in

Create the following symbolic link in /opt/sqlanywhere11/lib32:

libdblib11_r.so -> libdblib11_r.so.1

Linux/Solaris The SQL Anywhere Console

Create the following symbolic link in /opt/sqlanywhere11/lib32:

libdblib11_r.so -> libdblib11_r.so.1

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:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblgen11.res</td>
<td>English</td>
</tr>
<tr>
<td>dblgde11_iso_1.res, dblgde11_utf8.res</td>
<td>German (Linux only)</td>
</tr>
<tr>
<td>dblgja11_eucjis.res, dblgja11_sjis.res, dblgja11_utf8.res</td>
<td>Japanese (Linux only)</td>
</tr>
<tr>
<td>dblgzh11_cp936.res, dblgzh11_eucgb.res, dblgzh11_utf8.res</td>
<td>Simplified Chinese (Linux only)</td>
</tr>
</tbody>
</table>

You must also add the context-sensitive help files for those other languages. The available translated files are as follows:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>scvwen600.jar</td>
<td>English</td>
</tr>
<tr>
<td>scvwde600.jar</td>
<td>German</td>
</tr>
<tr>
<td>scvwfr600.jar</td>
<td>French</td>
</tr>
<tr>
<td>scvwja600.jar</td>
<td>Japanese</td>
</tr>
<tr>
<td>scvwzh600.jar</td>
<td>Simplified Chinese</td>
</tr>
</tbody>
</table>

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, but you have the flexibility to do this however is most appropriate for your application.

1. Set the PATH to include the following:
   
   `/opt/sqlanywhere11/bin32`
   
   (whichever is appropriate).

2. Set LD_LIBRARY_PATH to include the following:
   
   For Linux:
   
   `/opt/sqlanywhere11/jre_1.6.0_linux_sun_i586/lib/i386/client`
   `/opt/sqlanywhere11/jre_1.6.0_linux_sun_i586/lib/i386`
   `/opt/sqlanywhere11/jre_1.6.0_linux_sun_i586/lib/i386/native_threads`
   
   For Solaris:
   
   `/opt/sqlanywhere11/jre_1.6.0_solaris_sun_sparc/lib/sparc/client`
   `/opt/sqlanywhere11/jre_1.6.0_solaris_sun_sparc/lib/sparc`
   `/opt/sqlanywhere11/jre_1.6.0_solaris_sun_sparc/lib/sparc/native_threads`

3. Set the following environment variable:
   
   `SQLANY11="/opt/sqlanywhere11"`

Step 4: Registering the Sybase Central plug-ins

This step involves the configuration of Sybase Central. If you are not installing Sybase Central, you can skip it.

Sybase Central requires a configuration file that lists the installed plug-ins. Your installer must create this file. Note that it contains full paths to several JAR files that may change depending on where the software is installed.

The file is called `.scRepository600`. For most Linux and Unix systems, it resides in the `/opt/sqlanywhere11/bin32` directory. For Mac OS X, it resides in the `/opt/sqlanywhere11/bin64` directory. It is a plain text file that contains some basic information about the plug-ins that Sybase Central should load.

The provider information for SQL Anywhere is created in the repository file using the following commands.

```
scjview -register "/opt/sqlanywhere11/java/sqlanywhere.jpr"
```

The contents of the `sqlanywhere.jpr` file looks like this (some entries have been split across multiple lines for display purposes). The `AdditionalClasspath` lines must be entered on a single line in the `.jpr` file.

```
PluginName=SQL Anywhere 11
PluginId=sqlanywhere1100
PluginClass=ianywhere.sa.plugin.SAPlugin
PluginFile=_opt\sqlanywhere11\_java\saplugin.jar
AdditionalClasspath=_opt\sqlanywhere11\_java\isql.jar:
\_opt\sqlanywhere11\_java\salib.jar:
\_opt\sqlanywhere11\_java\JComponents1101.jar:
```
The `sqlanywhere.jpr` file was created in the `java` folder of the SQL Anywhere installation when you originally installed SQL Anywhere. Use it as the model for the `.jpr` file that you must create as part of the install process. There are also versions of this file for MobiLink and QAnywhere called `mobilink.jpr` and `qanywhere.jpr` respectively. They are also located in the `java` folder.

Here is a sample `.scRepository600` 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: 6.0.0.1154
# Fri Feb 23 13:09:14 EST 2007
# SCRepositoryInfo/Version=4
# Providers/sqlanywhere1100/Version=11.0.1.1297
Providers/sqlanywhere1100/UseClassLoader=true
Providers/sqlanywhere1100/ClassLoaderId=SA1100
Providers/sqlanywhere1100/Classpath=
    \_opt\sqlanywhere11\java\saplugin.jar
Providers/sqlanywhere1100/Name=SQL Anywhere 11
Providers/sqlanywhere1100/AdditionalClasspath=
    \_opt\sqlanywhere11\java\isql.jar:
    \_opt\sqlanywhere11\java\salib.jar:
    \_opt\sqlanywhere11\java\JComponents1101.jar:
    \_opt\sqlanywhere11\java\jlogon.jar:
    \_opt\sqlanywhere11\java\debugger.jar:
    \_opt\sqlanywhere11\java\jodbc.jar
Providers/sqlanywhere1100/Provider=ianywhere.sa.plugin.SAPlugin
Providers/sqlanywhere1100/ProviderId=sqlanywhere1100
Providers/sqlanywhere1100/InitialLoadOrder=0
#
```

**Notes**

- Your installer should create a file similar to this one using the techniques described above. The only changes required are the fully-qualified paths to the JAR files in the Classpath and AdditionalClasspath lines.
- The AdditionalClasspath lines shown above have wrapped to take up additional lines. They must be on a single line in the `.scRepository600` file.
- Forward slash characters (`/`) are represented with an escape sequence of `\` in the `.scRepository600` file.
- The first line indicates the version of the `.scRepository600` file.
- The lines beginning with `#` are comments.

For more information about deploying databases and database applications, see “Deploying databases and applications” on page 1017.
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 11 Demo is created in the .scRepository600 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 11 Demo connection profile. Use this as a model for creating your own connection profiles.

```
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Name" "SQL Anywhere 11 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/FirstTimeStart" "false"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Description" "Suitable Description"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/ProviderId" "sqlanywhere1100"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Provider" "SQL Anywhere 11"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileSettings" "DSN\eSQL^0020Anywhere^002011^0020Demo;UID \eDBA;PWD\e35c624d517fb"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileName" "SQL Anywhere 11 Demo"
scjview -write "ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileType" "SQL Anywhere"
```

The connection profile strings and values can be extracted from the .scRepository600 file. Define a connection profile using Sybase Central and then look at the .scRepository600 file for the corresponding lines.

Here is a portion of the .scRepository600 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: 6.0.0.1154
#  Fri Feb 23 13:09:14 EST 2007
#
ConnectionProfiles/SQL Anywhere 11 Demo/Name=SQL Anywhere 11 Demo
ConnectionProfiles/SQL Anywhere 11 Demo/FirstTimeStart=false
ConnectionProfiles/SQL Anywhere 11 Demo/Description=Suitable Description
ConnectionProfiles/SQL Anywhere 11 Demo/ProviderId=sqlanywhere1100
ConnectionProfiles/SQL Anywhere 11 Demo/Provider=SQL Anywhere 11
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileSettings=DSN\eSQL^0020Anywhere^002011^0020Demo;UID \eDBA;PWD\e35c624d517fb
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileName=SQL Anywhere 11 Demo
ConnectionProfiles/SQL Anywhere 11 Demo/Data/ConnectionProfileType=SQL Anywhere
```
Configuring the administration tools

You can control which features are shown or enabled by the administration tools. Control is done through an 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 11\java). If the file is not found, default values are used. Also, defaults are used for values that are missing from OEM.ini.

Here is a sample OEM.ini file:

```ini
[errors]
# reportErrors type is boolean, default = true
reportErrors=true

[updates]
# checkForUpdates type is boolean, default = true
checkForUpdates=true

[dbisql]
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.

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.

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 it at their discretion.

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 All Statement(s), then it is disabled also. Therefore, you might want to set the Execute toolbar button to Execute Selected Statement(s) 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].

Preventing users from changing Interactive SQL option settings

In the [dbisql] section of the OEM.ini file, you can lock the settings of Interactive SQL options so that users cannot change them. The option names are case sensitive. For some options, you can specify whether the setting is locked for SQL Anywhere databases or UltraLite databases. If you do not specify the database type, then the setting is locked for all databases. The following is an example:

```ini
[dbisql]
lockedPreferences=autoCommit
```
To lock the option setting for only one type of database, prefix `lockedPreferences` with the name of the database type (SQLAnywhere or UltraLite) followed by a period.

For example, if you want to lock `autoCommit` for SQL Anywhere databases, but not UltraLite, you would add the following line:

```plaintext
[dbisql]
SQLAnywhere.lockedPreferences=autoCommit
```

You can prevent users from changing the following Interactive SQL option settings (SQLAnywhere/UltraLite indicates that these options can be locked for a specific database type):

- **autoCommit (SQLAnywhere/UltraLite)** Prevents the user from customizing the Commit After Every Statement option. See “auto_commit option [Interactive SQL]” [SQL Anywhere Server - Database Administration].
- **autoRefetch (SQLAnywhere/UltraLite)** Prevents the user from customizing the Automatically Refetch Results option. See “auto_refetch option [Interactive SQL]” [SQL Anywhere Server - Database Administration].
- **commitOnExit (SQLAnywhere/UltraLite)** Prevents the user from customizing the Commit On Exit Or Disconnect option. See “commit_on_exit option [Interactive SQL]” [SQL Anywhere Server - Database Administration].
- **disableResultsEditing (SQLAnywhere/UltraLite)** Prevents the user from customizing the Disable Editing option.
- **executeToolbarButtonSemantics** Prevents the user from customizing the behavior of the Execute toolbar button. See “Configuring the Execute Statements toolbar button” [SQL Anywhere Server - Database Administration].
- **fastLauncherEnabled** Prevents the user from customizing the fast launcher option.
- **maximumDisplayedRows (SQLAnywhere/UltraLite)** Prevents the user from customizing the Maximum Number Of Rows To Display option. “isql_maximum_displayed_rows option [Interactive SQL]” [SQL Anywhere Server - Database Administration].
- **showMultipleResultSets (SQLAnywhere/UltraLite)** Prevents the user from customizing the Show Only The First Result Set or Show All Result Sets options. See “isql_show_multiple_result_sets [Interactive SQL]” [SQL Anywhere Server - Database Administration].
- **showResultsForAllStatements (SQLAnywhere/UltraLite)** Prevents the user from customizing the Show Results From The Last Statement or Show Results From Each Statement options.

**Deploying dbisqlc**

If your customer application is running on computers with limited resources, you may want to deploy the dbisqlc executable instead of Interactive SQL (dbisql). However, you should note that dbisqlc does not contain all the features of Interactive SQL and compatibility between the two is not guaranteed. Also, dbisqlc is deprecated because no new features are being added to it; there are currently no plans to remove it from the product.

The dbisqlc executable requires the standard embedded SQL client-side libraries.
For more information about dbisqlc, see “dbisqlc utility (deprecated)” [SQL Anywhere Server - Database Administration].

For more information about Interactive SQL (dbisql), see “Interactive SQL utility (dbisql)” [SQL Anywhere Server - Database Administration].
Deploying database servers

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 “Using the Deployment Wizard” on page 1023.

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.

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbeng11.exe</td>
<td>dbeng11</td>
<td>dbeng11</td>
</tr>
<tr>
<td>dbeng11.lic</td>
<td>dbeng11.lic</td>
<td>dbeng11.lic</td>
</tr>
<tr>
<td>dbsrv11.exe</td>
<td>dbsrv11</td>
<td>dbsrv11</td>
</tr>
<tr>
<td>dbsrv11.lic</td>
<td>dbsrv11.lic</td>
<td>dbsrv11.lic</td>
</tr>
<tr>
<td>dbserv11.dll</td>
<td>libdbserv11_r.so, libdbtasks11_r.so</td>
<td>libdbserv11_r.dylib, libdbtasks11_r.dylib</td>
</tr>
<tr>
<td>dbscript11.dll</td>
<td>libdbscript11_r.so</td>
<td>libdbscript11_r.dylib</td>
</tr>
<tr>
<td>dbghelp.dll</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>dbctrs11.dll</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>dbextf.dll</td>
<td>libdbextf.so 1</td>
<td>libdbextf.dylib 1</td>
</tr>
<tr>
<td>dbicu11.dll 2</td>
<td>libdbicu11_r.so 2</td>
<td>libdbicu11_r.dylib 2</td>
</tr>
<tr>
<td>dbicudt11.dll 2 3</td>
<td>libdbicudt11.so 2</td>
<td>libdbicudt11.dylib 2</td>
</tr>
<tr>
<td>sqlany.cvf</td>
<td>sqlany.cvf</td>
<td>sqlany.cvf</td>
</tr>
<tr>
<td>dbrsakp11.dll 4</td>
<td>libdbrsakp11_r.so 4</td>
<td>libdbrsakp11_r.dylib 4</td>
</tr>
<tr>
<td>dbodbc11.dll 5</td>
<td>libdbodbc11.so 5</td>
<td>libdbodbc11.dylib 5</td>
</tr>
<tr>
<td>N/A</td>
<td>libdbodbc11_n.so 5</td>
<td>libdbodbc11_n.dylib 5</td>
</tr>
<tr>
<td>N/A</td>
<td>libdbodbc11_r.so 5</td>
<td>libdbodbc11_r.dylib 5</td>
</tr>
<tr>
<td>dbjodbc11.dll 5</td>
<td>libdbjodbc11.so 5</td>
<td>libdbjodbc11.dylib 5</td>
</tr>
</tbody>
</table>
Before continuing, you should ensure that you have the following files:

### Windows

<table>
<thead>
<tr>
<th>File</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>java\jconn3.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\jodbc.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\sajvm.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\cis.zip</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libdbcis11.dll</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libsybbr.dll</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>java\jconn3.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\jodbc.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\sajvm.jar</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>java\cis.zip</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libdbcis11.so</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libdbcis11.dylib</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libsybbr.so</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
<tr>
<td>libsybbr.dylib</td>
<td>Windows, Linux, Unix, Mac OS X</td>
</tr>
</tbody>
</table>

1. Required only if using system extended stored procedures and functions (xp_ *).
2. Required only if the database character set is multi-byte or if the UCA collation sequence is used.
3. On Windows Mobile, the file name to deploy is called dbicudt11.dat.
4. Required only for encrypted TDS connections.
5. Required only if using Java in the database.
6. Required only if using Java in the database and remote data access.
7. Required only if using remote data access.
8. Required only for archive backups.

### Notes

- Depending on your situation, you should choose whether to deploy the personal database server (dbeng11) or the network database server (dbsrv11).
- You must include the separate corresponding license file (dbeng11.lic or dbsrv11.lic) when deploying a database server. The license files are located in the same directory as the server executables.
- The table above shows files with the designation [xx]. 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 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 dbackup.

For information about deploying utilities, see “Deploying administration tools” on page 1048.

### Windows Registry entries

To ensure that messages written by the server to the Event Log on Windows are formatted correctly, create the following registry key.

```
HKEY_LOCAL_MACHINE\SYSTEM\
```
Within this key, add a REG_SZ value named EventMessageFile and assign it the data value of the fully qualified location of `dblgen11.dll`, for example, `C:\Program Files\SQL Anywhere 11\bin32\dblgen11.dll`. The English language DLL, `dblgen11.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 11.0]
"EventMessageFile"="c:\sa11\bin32\dblgen11.dll"
```

For the 64-bit version of the server, the registry key is `SQLANY64 11.0`.

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 11.0 Admin
```

Within this key, add a REG_SZ value named EventMessageFile and assign it the data value of the fully qualified location of `dblgen11.dll`, for example, `C:\Program Files\SQL Anywhere 11\bin32\dblgen11.dll`. The English language DLL, `dblgen11.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 11.0 Admin]
"EventMessageFile"="c:\sa11\bin32\dblgen11.dll"
```

For the 64-bit version of the server, the registry key is `SQLANY64 11.0 Admin`.

You can suppress Windows event log entries by setting up a registry key. The registry key is:

```
Software\Sybase\SQL Anywhere\11.0\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.
Registering DLLs on Windows

When deploying SQL Anywhere, there are DLL files that must be registered for SQL Anywhere to function properly. Note that for Windows Vista or later versions of Windows, you must include the SQL Anywhere elevated operations agent (db elevate 11.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.

To register a DLL

1. Open a command prompt.
2. Change to the directory where the DLL provider is installed.
3. Enter the following command to register the provider (in this example, the OLE DB provider is registered):
   
   regsvr32 dboledb11.dll

The following table lists the DLLs that must be registered when deploying SQL Anywhere:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbctrs11.dll</td>
<td>The SQL Anywhere Performance Monitor counters.</td>
</tr>
<tr>
<td>dbmlsynccom.dll</td>
<td>The Dbmlsync Integration Component (non-visual component).</td>
</tr>
<tr>
<td>dbmlsynccomg.dll</td>
<td>The Dbmlsync Integration Component (visual component).</td>
</tr>
<tr>
<td>dbodbc11.dll</td>
<td>The SQL Anywhere ODBC driver.</td>
</tr>
<tr>
<td>dboledb11.dll</td>
<td>The SQL Anywhere OLE DB provider.</td>
</tr>
<tr>
<td>dboledba11.dll</td>
<td>The SQL Anywhere OLE DB provider schema assist module.</td>
</tr>
<tr>
<td>Windows\system32\msxml4.dll</td>
<td>The Microsoft XML Parser.</td>
</tr>
</tbody>
</table>

Deploying databases

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.

For information about extracting databases, see “Extracting remote databases” [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]
- “Choosing collations” [SQL Anywhere Server - Database Administration]

Deploying databases 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 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.
# Deploying external environment support

The following tables summarize the components that must be deployed to support external calls in SQL Anywhere.

## ESQL/ODBC external calls

<table>
<thead>
<tr>
<th>Component</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESQL and ODBC launcher</td>
<td>dbexternc11.exe</td>
<td>dbexternc11</td>
<td>dbexternc11</td>
</tr>
<tr>
<td>Bridge</td>
<td>dbextenv11.dll</td>
<td>libdbextenv11_r.so</td>
<td>libdbextenv11_r.dylib</td>
</tr>
<tr>
<td>SQL Anywhere C API runtime</td>
<td>dbcapi.dll</td>
<td>libdbcapi_r.so</td>
<td>libdbcapi_r.dylib</td>
</tr>
</tbody>
</table>

For additional files required by embedded SQL applications, see “Deploying embedded SQL clients” on page 1044.

For additional files required by ODBC applications, see “Deploying ODBC clients” on page 1037.

## Java external calls

<table>
<thead>
<tr>
<th>Component</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java installation (third-party)</td>
<td>java.exe</td>
<td>java</td>
<td>java</td>
</tr>
<tr>
<td>Launcher</td>
<td>sajvm.jar</td>
<td>sajvm.jar</td>
<td>sajvm.jar</td>
</tr>
<tr>
<td>iAnywhere JDBC driver (server-side calls)</td>
<td>dbjodbc11.dll</td>
<td>libdbjodbc11.so</td>
<td>libdbjodbc11.dylib</td>
</tr>
</tbody>
</table>

For additional files required by JDBC applications, see “Deploying JDBC clients” on page 1046.

## .NET CLR external calls

<table>
<thead>
<tr>
<th>Component</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET 2.0 or higher</td>
<td>(from Microsoft)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>.NET CLR bridge</td>
<td>dbextclr11.exe</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Bridge</td>
<td>dbextenv11.dll</td>
<td>libdbextenv11_r.so</td>
<td>libdbextenv11_r.dylib</td>
</tr>
<tr>
<td>.NET CLR support</td>
<td>dbclrenv11.dll</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Perl external calls

<table>
<thead>
<tr>
<th>Component</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl installation (third-party)</td>
<td>perl.exe</td>
<td>perl</td>
<td>perl</td>
</tr>
<tr>
<td>Perl launcher</td>
<td>perlenv.pl</td>
<td>perlenv.pl</td>
<td>perlenv.pl</td>
</tr>
<tr>
<td>Bridge</td>
<td>dbextenv11.dll</td>
<td>libdbextenv11_r.so</td>
<td>libdbextenv11_r.dylib</td>
</tr>
<tr>
<td>SQL Anywhere C API runtime</td>
<td>dbcapi.dll</td>
<td>libdbcapi_r.so</td>
<td>libdbcapi_r.dylib</td>
</tr>
</tbody>
</table>

For additional files required by Perl applications, also “Introduction to DBD::SQLAnywhere” on page 722.

## PHP external calls

<table>
<thead>
<tr>
<th>Component</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP installation (third-party)</td>
<td>php.exe</td>
<td>php</td>
<td>php</td>
</tr>
<tr>
<td>PHP launcher</td>
<td>phpenv.php</td>
<td>phpenv.php</td>
<td>phpenv.php</td>
</tr>
<tr>
<td>Bridge</td>
<td>dbextenv11.dll</td>
<td>libdbextenv11_r.so</td>
<td>libdbextenv11_r.dylib</td>
</tr>
<tr>
<td>PHP 5.1.x external calls</td>
<td>php-5.1.1-{1-6}_sqlanywhere_extenv11.dll</td>
<td>php-5.1.1-{1-6}_sqlanywhere_extenv11_r.so or build from source code</td>
<td>Build from source code</td>
</tr>
<tr>
<td>PHP 5.2.x external calls</td>
<td>php-5.2.0-{0-6}_sqlanywhere_extenv11.dll</td>
<td>php-5.2.0-{0-6}_sqlanywhere_extenv11_r.so or build from source code</td>
<td>Build from source code</td>
</tr>
<tr>
<td>SQL Anywhere C API runtime</td>
<td>dbcapi.dll</td>
<td>libdbcapi_r.so</td>
<td>libdbcapi_r.dylib</td>
</tr>
<tr>
<td>DLBLIB (threaded)</td>
<td>dplib11.dll</td>
<td>libdplib11_r.so</td>
<td>libdplib11_r.dylib</td>
</tr>
</tbody>
</table>

For additional files required by PHP applications, see “Introduction to the SQL Anywhere PHP module” on page 740.
Deploying security

The following table summarizes the components that support security features in SQL Anywhere.

<table>
<thead>
<tr>
<th>Security option</th>
<th>Security type</th>
<th>Included in module</th>
<th>Licensable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database encryption</td>
<td>AES</td>
<td>dbserv11.dll</td>
<td>included¹</td>
</tr>
<tr>
<td></td>
<td></td>
<td>libdbserv11_r.so</td>
<td></td>
</tr>
<tr>
<td>Database encryption</td>
<td>FIPS-approved AES</td>
<td>dbfips11.dll</td>
<td>separately licensed²</td>
</tr>
<tr>
<td>Transport layer security</td>
<td>RSA</td>
<td>dbrsa11.dll</td>
<td>included¹</td>
</tr>
<tr>
<td></td>
<td></td>
<td>libdbrsa11.so</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>libdbrsa11.dylib</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>libdbrsa11_r.dylib</td>
<td></td>
</tr>
<tr>
<td>Transport layer security</td>
<td>FIPS-approved RSA</td>
<td>dbfips11.dll, sbgse2.dll</td>
<td>separately licensed²</td>
</tr>
<tr>
<td>Transport layer security</td>
<td>ECC</td>
<td>dbecc11.dll</td>
<td>separately licensed²</td>
</tr>
<tr>
<td></td>
<td></td>
<td>libdbecc11.so</td>
<td></td>
</tr>
</tbody>
</table>

¹ AES and RSA strong encryption are included with SQL Anywhere and do not require a separate license, but these libraries are not FIPS-certified.

² The software for strong encryption using ECC or FIPS-certified technology must be ordered separately.
Deploying embedded database applications

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.
  
  For information about deploying client applications, see “Deploying client applications” on page 1029.

- **Database server** The SQL Anywhere personal database server.
  
  For information about deploying database servers, see “Deploying database servers” on page 1074.

- **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.

Deploying personal servers

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 (`dblgen11.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.

Deploying database utilities

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:

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database tools library</td>
<td><code>dbtool11.dll</code></td>
<td><code>libdbtool11_r.so, libdbtasks11_r.so</code></td>
<td><code>libdbtool11_r.dylib, libdbtasks11_r.dylib</code></td>
</tr>
<tr>
<td>Interface Library</td>
<td><code>dblib11.dll</code></td>
<td><code>libdblib11_r.so</code></td>
<td><code>libdblib11_r.dylib</code></td>
</tr>
</tbody>
</table>
## Deploying unload support for pre 10.0.0 databases

If, in your application, you need the ability to convert older databases to the version 11 format, then you need the database unload utility, dbunload, together with the following additional files:

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
<th>Linux / Unix</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unload support for pre-10.0 databases</td>
<td>dbunlspt.exe</td>
<td>dbunlspt</td>
<td>dbunlspt</td>
</tr>
</tbody>
</table>

The table above shows files with the designation [xx]. 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

<table>
<thead>
<tr>
<th>dbusde.dll</th>
<th>German</th>
</tr>
</thead>
</table>

Notes

- The table above shows files with the designation [xx]. 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 non-multi-threaded applications on Linux and Unix, you can use `libdbtasks11.so` and `libdblib11.so`.

- For non-multi-threaded applications on Mac OS X, you can use `libdbtasks11.dylib` and `libdblib11.dylib`.

- The pre-10 physical store library is required by some utilities (dblog, dbtran, dberase) to access pre-version 10.0.0 log files. If you are not deploying these utilities, then you do not require this library.

- The personal database server (dbeng11) 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 “Deploying database servers” on page 1074.

- The dbunload utility may require the contents of the `scripts` directory to be present.
### Deploying databases and applications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbusen.dll</code></td>
<td>English</td>
</tr>
<tr>
<td><code>dbuses.dll</code></td>
<td>Spanish</td>
</tr>
<tr>
<td><code>dbusfr.dll</code></td>
<td>French</td>
</tr>
<tr>
<td><code>busit.dll</code></td>
<td>Italian</td>
</tr>
<tr>
<td><code>dbusja.dll</code></td>
<td>Japanese</td>
</tr>
<tr>
<td><code>dbusko.dll</code></td>
<td>Korean</td>
</tr>
<tr>
<td><code>dbuslt.dll</code></td>
<td>Lithuanian</td>
</tr>
<tr>
<td><code>dbuspl.dll</code></td>
<td>Polish</td>
</tr>
<tr>
<td><code>busp.dll</code></td>
<td>Portuguese</td>
</tr>
<tr>
<td><code>busru.dll</code></td>
<td>Russian</td>
</tr>
<tr>
<td><code>busw.dll</code></td>
<td>Traditional Chinese</td>
</tr>
<tr>
<td><code>busuk.dll</code></td>
<td>Ukrainian</td>
</tr>
<tr>
<td><code>buszh.dll</code></td>
<td>Simplified Chinese</td>
</tr>
</tbody>
</table>

#### Linux message files

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbusde_iso_1.res, dbusde_utf8.res, dbusen.res</code></td>
<td>German</td>
</tr>
<tr>
<td><code>dbusen.res</code></td>
<td>English</td>
</tr>
<tr>
<td><code>dbusja_eucjis.res, dbusja_sjis.res, dbusja_utf8.res</code></td>
<td>Japanese</td>
</tr>
<tr>
<td><code>dbuszh_cp936.res, dbuszh_eucgb.res, dbuszh_utf8.res</code></td>
<td>Chinese</td>
</tr>
</tbody>
</table>

These files are included with localized versions of SQL Anywhere.

In addition to these files, you also need the files described in “Deploying database utilities” on page 1082.

### Deploying SQL Remote

If you are deploying the SQL Remote Message Agent, you need to include the following files:

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows</th>
<th>Linux / Solaris</th>
<th>Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Agent</td>
<td><code>dbremote.exe</code></td>
<td><code>dbremote</code></td>
<td><code>dbremote</code></td>
</tr>
<tr>
<td>Description</td>
<td>Windows</td>
<td>Linux / Solaris</td>
<td>Mac OS X</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------</td>
<td>-------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Encoding/decoding library</td>
<td>dbencod11.dll</td>
<td>libdbencod11_r.so.1</td>
<td>libdbencod11_r.dylib</td>
</tr>
<tr>
<td>FILE message link library</td>
<td>dbfile11.dll</td>
<td>libdbfile11_r.so.1</td>
<td>libdbfile11_r.dylib</td>
</tr>
<tr>
<td>FTP message link library 1</td>
<td>dbftp11.dll</td>
<td>libdbftp11_r.so.1</td>
<td>libdbftp11_r.dylib</td>
</tr>
<tr>
<td>Interface Library</td>
<td>dlib11.dll</td>
<td>libdlib11_r.so.1</td>
<td>libdlib11_r.dylib</td>
</tr>
<tr>
<td>SMTP message link library 1</td>
<td>dbsmtp11.dll</td>
<td>libdbsmtp11_r.so.1</td>
<td>libdbsmtp11_r.dylib</td>
</tr>
<tr>
<td>Database tools library</td>
<td>dbtool11.dll</td>
<td>libdbtool11_r.so.1</td>
<td>libdbtool11_r.dylib</td>
</tr>
<tr>
<td>Thread support library</td>
<td>N/A</td>
<td>libdbtasks11_r.so.1</td>
<td>libdbtasks11_r.dylib</td>
</tr>
</tbody>
</table>

1 Only deploy the library for the message link you are using.

The table above shows files with the designation [xx]. 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.

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.
Glossary
Adaptive Server Anywhere (ASA)

The relational database server component of SQL Anywhere Studio, intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. In version 10.0.0, Adaptive Server Anywhere was renamed SQL Anywhere Server, and SQL Anywhere Studio was renamed SQL Anywhere.

See also: “SQL Anywhere” on page 1113.

agent ID

See also: “client message store ID” on page 1091.

article

In MobiLink or SQL Remote, an article is a database object that represents a whole table, or a subset of the columns and rows in a table. Articles are grouped together in a publication.

See also:

● “replication” on page 1111
● “publication” on page 1108

atomic transaction

A transaction that is guaranteed to complete successfully or not at all. If an error prevents part of an atomic transaction from completing, the transaction is rolled back to prevent the database from being left in an inconsistent state.

base table

Permanent tables for data. Tables are sometimes called base tables to distinguish them from temporary tables and views.

See also:

● “temporary table” on page 1115
● “view” on page 1117
bit array
A bit array is a type of array data structure that is used for efficient storage of a sequence of bits. A bit array is similar to a character string, except that the individual pieces are 0s (zeros) and 1s (ones) instead of characters. Bit arrays are typically used to hold a string of Boolean values.

business rule
A guideline based on real-world requirements. Business rules are typically implemented through check constraints, user-defined data types, and the appropriate use of transactions.

See also:
- “constraint” on page 1093
- “user-defined data type” on page 1117

carrier
A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

See also: “server-initiated synchronization” on page 1112.

character set
A character set is a set of symbols, including letters, digits, spaces, and other symbols. An example of a character set is ISO-8859-1, also known as Latin1.

See also:
- “code page” on page 1091
- “encoding” on page 1097
- “collation” on page 1091

check constraint
A restriction that enforces specified conditions on a column or set of columns.

See also:
- “constraint” on page 1093
- “foreign key constraint” on page 1098
- “primary key constraint” on page 1108
- “unique constraint” on page 1116

checkpoint
The point at which all changes to the database are saved to the database file. At other times, committed changes are saved only to the transaction log.
checksum

The calculated number of bits of a database page that is recorded with the database page itself. The checksum allows the database management system to validate the integrity of the page by ensuring that the numbers match as the page is being written to disk. If the counts match, it’s assumed that page was successfully written.

client message store

In QAnywhere, a SQL Anywhere database on the remote device that stores messages.

client message store ID

In QAnywhere, a MobiLink remote ID that uniquely identifies a client message store.

client/server

A software architecture where one application (the client) obtains information from and sends information to another application (the server). The two applications often reside on different computers connected by a network.

code page

A code page is an encoding that maps characters of a character set to numeric representations, typically an integer between 0 and 255. An example of a code page is Windows code page 1252. For the purposes of this documentation, code page and encoding are interchangeable terms.

See also:

- “character set” on page 1090
- “encoding” on page 1097
- “collation” on page 1091

collation

A combination of a character set and a sort order that defines the properties of text in the database. For SQL Anywhere databases, the default collation is determined by the operating system and language on which the server is running; for example, the default collation on English Windows systems is 1252LATIN1. A collation, also called a collating sequence, is used for comparing and sorting strings.

See also:

- “character set” on page 1090
- “code page” on page 1091
- “encoding” on page 1097

command file

A text file containing SQL statements. Command files can be built manually, or they can be built automatically by database utilities. The dbunload utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.
communication stream

In MobiLink, the network protocol used for communication between the MobiLink client and the MobiLink server.

concurrency

The simultaneous execution of two or more independent, and possibly competing, processes. SQL Anywhere automatically uses locking to isolate transactions and ensure that each concurrent application sees a consistent set of data.

See also:

- “transaction” on page 1115
- “isolation level” on page 1101

conflict resolution

In MobiLink, conflict resolution is logic that specifies what to do when two users modify the same row on different remote databases.

connection ID

A unique number that identifies a given connection between a client application and the database. You can determine the current connection ID using the following SQL statement:

```
SELECT CONNECTIONPROPERTY('Number');
```

connection-initiated synchronization

A form of MobiLink server-initiated synchronization in which synchronization is initiated when there are changes to connectivity.

See also: “server-initiated synchronization” on page 1112.

connection profile

A set of parameters that are required to connect to a database, such as user name, password, and server name, that is stored and used as a convenience.

consolidated database

In distributed database environments, a database that stores the master copy of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of the data.

See also:

- “synchronization” on page 1115
- “replication” on page 1111
constraint

A restriction on the values contained in a particular database object, such as a table or column. For example, a column may have a uniqueness constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to data in some other table.

See also:

- “check constraint” on page 1090
- “foreign key constraint” on page 1098
- “primary key constraint” on page 1108
- “unique constraint” on page 1116

contention

The act of competing for resources. For example, in database terms, two or more users trying to edit the same row of a database contend for the rights to edit that row.

correlation name

The name of a table or view that is used in the FROM clause of a query—either its original name, or an alternate name, that is defined in the FROM clause.

creator ID

In UltraLite Palm OS applications, an ID that is assigned when the application is created.

cursor

A named linkage to a result set, used to access and update rows from a programming interface. In SQL Anywhere, cursors support forward and backward movement through the query results. Cursors consist of two parts: the cursor result set, typically defined by a SELECT statement; and the cursor position.

See also:

- “cursor result set” on page 1093
- “cursor position” on page 1093

cursor position

A pointer to one row within the cursor result set.

See also:

- “cursor” on page 1093
- “cursor result set” on page 1093

cursor result set

The set of rows resulting from a query that is associated with a cursor.
See also:

- “cursor” on page 1093
- “cursor position” on page 1093

**data cube**

A multi-dimensional result set with each dimension reflecting a different way to group and sort the same results. Data cubes provide complex information about data that would otherwise require self-join queries and correlated subqueries. Data cubes are a part of OLAP functionality.

**data definition language (DDL)**

The subset of SQL statements for defining the structure of data in the database. DDL statements create, modify, and remove database objects, such as tables and users.

**data manipulation language (DML)**

The subset of SQL statements for manipulating data in the database. DML statements retrieve, insert, update, and delete data in the database.

**data type**

The format of data, such as CHAR or NUMERIC. In the ANSI SQL standard, data types can also include a restriction on size, character set, and collation.

See also: “domain” on page 1096.

**database**

A collection of tables that are related by primary and foreign keys. The tables hold the information in the database. The tables and keys together define the structure of the database. A database management system accesses this information.

See also:

- “foreign key” on page 1098
- “primary key” on page 1108
- “database management system (DBMS)” on page 1095
- “relational database management system (RDBMS)” on page 1110

**database administrator (DBA)**

The user with the permissions required to maintain the database. The DBA is generally responsible for all changes to a database schema, and for managing users and groups. The role of database administrator is automatically built into databases as user ID DBA with password sql.
**database connection**

A communication channel between a client application and the database. A valid user ID and password are required to establish a connection. The privileges granted to the user ID determine the actions that can be carried out during the connection.

**database file**

A database is held in one or more database files. There is an initial file, and subsequent files are called dbspaces. Each table, including its indexes, must be contained within a single database file.

See also: “dbspace” on page 1096.

**database management system (DBMS)**

A collection of programs that allow you to create and use databases.

See also: “relational database management system (RDBMS)” on page 1110.

**database name**

The name given to a database when it is loaded by a server. The default database name is the root of the initial database file.

See also: “database file” on page 1095.

**database object**

A component of a database that contains or receives information. Tables, indexes, views, procedures, and triggers are database objects.

**database owner (dbo)**

A special user that owns the system objects not owned by SYS.

See also:

- “database administrator (DBA)” on page 1094
- “SYS” on page 1115

**database server**

A computer program that regulates all access to information in a database. SQL Anywhere provides two types of servers: network servers and personal servers.

**DBA authority**

The level of permission that enables a user to do administrative activity in the database. The DBA user has DBA authority by default.

See also: “database administrator (DBA)” on page 1094.
dbspace

An additional database file that creates more space for data. A database can be held in up to 13 separate files (an initial file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

See also: “database file” on page 1095.

deadlock

A state where a set of transactions arrives at a place where none can proceed.

device tracking

In MobiLink server-initiated synchronization, functionality that allows you to address messages using the MobiLink user name that identifies a device.

See also: “server-initiated synchronization” on page 1112.

direct row handling

In MobiLink, a way to synchronize table data to sources other than the MobiLink-supported consolidated databases. You can implement both uploads and downloads with direct row handling.

See also:

- “consolidated database” on page 1092
- “SQL-based synchronization” on page 1113

domain

Aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are pre-defined in SQL Anywhere. Also called user-defined data type.

See also: “data type” on page 1094.

download

The stage in synchronization where data is transferred from the consolidated database to a remote database.

dynamic SQL

SQL that is generated programmatically by your program before it is executed. UltraLite dynamic SQL is a variant designed for small-footprint devices.

EBF

Express Bug Fix. An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. Some testing has been performed on the software, but the software has not
undergone full testing. You should not distribute these files with your application unless you have verified the suitability of the software yourself.

**embedded SQL**

A programming interface for C programs. SQL Anywhere embedded SQL is an implementation of the ANSI and IBM standard.

**encoding**

Also known as character encoding, an encoding is a method by which each character in a character set is mapped onto one or more bytes of information, typically represented as a hexadecimal number. An example of an encoding is UTF-8.

See also:

- “character set” on page 1090
- “code page” on page 1091
- “collation” on page 1091

**event model**

In MobiLink, the sequence of events that make up a synchronization, such as begin_synchronization and download_cursor. Events are invoked if a script is created for them.

**external login**

An alternate login name and password used when communicating with a remote server. By default, SQL Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords used when communicating with a remote server.

**extraction**

In SQL Remote replication, the act of unloading the appropriate structure and data from the consolidated database. This information is used to initialize the remote database.

See also: “replication” on page 1111.

**failover**

Switching to a redundant or standby server, system, or network on failure or unplanned termination of the active server, system, or network. Failover happens automatically.

**FILE**

In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system.

See also: “replication” on page 1111.
file-based download

In MobiLink, a way to synchronize data in which downloads are distributed as files, allowing offline distribution of synchronization changes.

file-definition database

In MobiLink, a SQL Anywhere database that is used for creating download files.

See also: “file-based download” on page 1098.

foreign key

One or more columns in a table that duplicate the primary key values in another table. Foreign keys establish relationships between tables.

See also:

● “primary key” on page 1108
● “foreign table” on page 1098

foreign key constraint

A restriction on a column or set of columns that specifies how the data in the table relates to the data in some other table. Imposing a foreign key constraint on a set of columns makes those columns the foreign key.

See also:

● “constraint” on page 1093
● “check constraint” on page 1090
● “primary key constraint” on page 1108
● “unique constraint” on page 1116

foreign table

The table containing the foreign key.

See also: “foreign key” on page 1098.

full backup

A backup of the entire database, and optionally, the transaction log. A full backup contains all the information in the database and provides protection in the event of a system or media failure.

See also: “incremental backup” on page 1100.

gateway

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about how to send messages for server-initiated synchronization.

See also: “server-initiated synchronization” on page 1112.
generated join condition

A restriction on join results that is automatically generated. There are two types: key and natural. Key joins are generated when you specify KEY JOIN or when you specify the keyword JOIN but do not use the keywords CROSS, NATURAL, or ON. For a key join, the generated join condition is based on foreign key relationships between tables. Natural joins are generated when you specify NATURAL JOIN; the generated join condition is based on common column names in the two tables.

See also:

- “join” on page 1102
- “join condition” on page 1102

generation number

In MobiLink, a mechanism for forcing remote databases to upload data before applying any more download files.

See also: “file-based download” on page 1098.

global temporary table

A type of temporary table for which data definitions are visible to all users until explicitly dropped. Global temporary tables let each user open their own identical instance of a table. By default, rows are deleted on commit, and rows are always deleted when the connection is ended.

See also:

- “temporary table” on page 1115
- “local temporary table” on page 1102

grant option

The level of permission that allows a user to grant permissions to other users.

hash

A hash is an index optimization that transforms index entries into keys. An index hash aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value, by including enough of the actual row data with its row ID.

histogram

The most important component of column statistics, histograms are a representation of data distribution. SQL Anywhere maintains histograms to provide the optimizer with statistical information about the distribution of values in columns.
iAnywhere JDBC driver

The iAnywhere JDBC driver provides a JDBC driver that has some performance benefits and feature benefits compared to the pure Java jConnect JDBC driver, but which is not a pure-Java solution. The iAnywhere JDBC driver is recommended in most cases.

See also:

● “JDBC” on page 1101
● “jConnect” on page 1101

identifier

A string of characters used to reference a database object, such as a table or column. An identifier may contain any character from A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign ($).

incremental backup

A backup of the transaction log only, typically used between full backups.

See also: “transaction log” on page 1115.

index

A sorted set of keys and pointers associated with one or more columns in a base table. An index on one or more columns of a table can improve performance.

InfoMaker

A reporting and data maintenance tool that lets you create sophisticated forms, reports, graphs, cross-tabs, and tables, and applications that use these reports as building blocks.

inner join

A join in which rows appear in the result set only if both tables satisfy the join condition. Inner joins are the default.

See also:

● “join” on page 1102
● “outer join” on page 1106

integrated login

A login feature that allows the same single user ID and password to be used for operating system logins, network logins, and database connections.
integrity

Adherence to rules that ensure that data is correct and accurate, and that the relational structure of the database is intact.

See also: “referential integrity” on page 1110.

Interactive SQL

A SQL Anywhere application that allows you to query and alter data in your database, and modify the structure of your database. Interactive SQL provides a pane for you to enter SQL statements, and panes that return information about how the query was processed and the result set.

isolation level

The degree to which operations in one transaction are visible to operations in other concurrent transactions. There are four isolation levels, numbered 0 through 3. Level 3 provides the highest level of isolation. Level 0 is the default setting. SQL Anywhere also supports three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot.

See also: “snapshot isolation” on page 1113.

JAR file

Java archive file. A compressed file format consisting of a collection of one or more packages used for Java applications. It includes all the resources necessary to install and run a Java program in a single compressed file.

Java class

The main structural unit of code in Java. It is a collection of procedures and variables grouped together because they all relate to a specific, identifiable category.

jConnect

A Java implementation of the JavaSoft JDBC standard. It provides Java developers with native database access in multi-tier and heterogeneous environments. However, the iAnywhere JDBC driver is the preferred JDBC driver for most cases.

See also:

- “JDBC” on page 1101
- “iAnywhere JDBC driver” on page 1100

JDBC

Java Database Connectivity. A SQL-language programming interface that allows Java applications to access relational data. The preferred JDBC driver is the iAnywhere JDBC driver.
See also:

- “jConnect” on page 1101
- “iAnywhere JDBC driver” on page 1100

**join**

A basic operation in a relational system that links the rows in two or more tables by comparing the values in specified columns.

**join condition**

A restriction that affects join results. You specify a join condition by inserting an ON clause or WHERE clause immediately after the join. In the case of natural and key joins, SQL Anywhere generates a join condition.

See also:

- “join” on page 1102
- “generated join condition” on page 1099

**join type**

SQL Anywhere provides four types of joins: cross join, key join, natural join, and joins using an ON clause.

See also: “join” on page 1102.

**light weight poller**

In MobiLink server-initiated synchronization, a device application that polls for push notifications from a MobiLink server.

See also: “server-initiated synchronization” on page 1112.

**Listener**

A program, dbdsn, that is used for MobiLink server-initiated synchronization. Listeners are installed on remote devices and configured to initiate actions on the device when they receive push notifications.

See also: “server-initiated synchronization” on page 1112.

**local temporary table**

A type of temporary table that exists only for the duration of a compound statement or until the end of the connection. Local temporary tables are useful when you need to load a set of data only once. By default, rows are deleted on commit.

See also:

- “temporary table” on page 1115
- “global temporary table” on page 1099
lock
A concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions. SQL Anywhere automatically applies locks to prevent two connections from changing the same data at the same time, and to prevent other connections from reading data that is in the process of being changed.

You control locking by setting the isolation level.

See also:
- “isolation level” on page 1101
- “Concurrency” on page 1092
- “integrity” on page 1101

log file
A log of transactions maintained by SQL Anywhere. The log file is used to ensure that the database is recoverable in the event of a system or media failure, to improve database performance, and to allow data replication using SQL Remote.

See also:
- “transaction log” on page 1115
- “transaction log mirror” on page 1116
- “full backup” on page 1098

logical index
A reference (pointer) to a physical index. There is no indexing structure stored on disk for a logical index.

LTM
Log Transfer Manager (LTM) also called Replication Agent. Used with Replication Server, the LTM is the program that reads a database transaction log and sends committed changes to Sybase Replication Server.

See: “Replication Server” on page 1111.

maintenance release
A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is major.minor.patch.build). Bug fixes and other changes are listed in the release notes for the upgrade.

materialized view
A materialized view is a view that has been computed and stored on disk. Materialized views have characteristics of both views (they are defined using a query specification), and of tables (they allow most table operations to be performed on them).
message log

A log where messages from an application such as a database server or MobiLink server can be stored. This information can also appear in a messages window or be logged to a file. The message log includes informational messages, errors, warnings, and messages from the MESSAGE statement.

message store

In QAnywhere, databases on the client and server device that store messages.

message system

In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. SQL Anywhere includes support for the following message systems: FILE, FTP, and SMTP.

message type

In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems.

metadata

Data about data. Metadata describes the nature and content of other data.

mirror log

See also: “transaction log mirror” on page 1116.
MobiLink

A session-based synchronization technology designed to synchronize UltraLite and SQL Anywhere remote databases with a consolidated database.

See also:

- “consolidated database” on page 1092
- “synchronization” on page 1115
- “UltraLite” on page 1116

MobiLink client

There are two kinds of MobiLink clients. For SQL Anywhere remote databases, the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.

MobiLink Monitor

A graphical tool for monitoring MobiLink synchronizations.

MobiLink server

The computer program that runs MobiLink synchronization, mlsrv11.

MobiLink system table

System tables that are required by MobiLink synchronization. They are installed by MobiLink setup scripts into the MobiLink consolidated database.

MobiLink user

A MobiLink user is used to connect to the MobiLink server. You create the MobiLink user on the remote database and register it in the consolidated database. MobiLink user names are entirely independent of database user names.

network protocol

The type of communication, such as TCP/IP or HTTP.

network server

A database server that accepts connections from computers sharing a common network.

See also: “personal server” on page 1107.

normalization

The refinement of a database schema to eliminate redundancy and improve organization according to rules based on relational database theory.
Notifier

A program that is used by MobiLink server-initiated synchronization. Notifiers are integrated into the MobiLink server. They check the consolidated database for push requests, and send push notifications.

See also:

- “server-initiated synchronization” on page 1112
- “Listener” on page 1102

object tree

In Sybase Central, the hierarchy of database objects. The top level of the object tree shows all products that your version of Sybase Central supports. Each product expands to reveal its own sub-tree of objects.

See also: “Sybase Central” on page 1114.

ODBC

Open Database Connectivity. A standard Windows interface to database management systems. ODBC is one of several interfaces supported by SQL Anywhere.

ODBC Administrator

A Microsoft program included with Windows operating systems for setting up ODBC data sources.

ODBC data source

A specification of the data a user wants to access via ODBC, and the information needed to get to that data.

outer join

A join that preserves all the rows in a table. SQL Anywhere supports left, right, and full outer joins. A left outer join preserves the rows in the table to the left of the join operator, and returns a null when a row in the right table does not satisfy the join condition. A full outer join preserves all the rows from both tables.

See also:

- “join” on page 1102
- “inner join” on page 1100

package

In Java, a collection of related classes.

parse tree

An algebraic representation of a query.

PDB

A Palm database file.
**performance statistic**

A value reflecting the performance of the database system. The CURRREAD statistic, for example, represents the number of file reads issued by the database server that have not yet completed.

**personal server**

A database server that runs on the same computer as the client application. A personal database server is typically used by a single user on a single computer, but it can support several concurrent connections from that user.

**physical index**

The actual indexing structure of an index, as it is stored on disk.

**plug-in module**

In Sybase Central, a way to access and administer a product. Plug-ins are usually installed and registered automatically with Sybase Central when you install the respective product. Typically, a plug-in appears as a top-level container, in the Sybase Central main window, using the name of the product itself; for example, SQL Anywhere.

See also: “Sybase Central” on page 1114.

**policy**

In QAnywhere, the way you specify when message transmission should occur.

**polling**

In MobiLink server-initiated synchronization, the way a light weight poller, such as the MobiLink Listener, requests push notifications from a Notifier.

See also: “server-initiated synchronization” on page 1112.

**PowerDesigner**

A database modeling application. PowerDesigner provides a structured approach to designing a database or data warehouse. SQL Anywhere includes the Physical Data Model component of PowerDesigner.

**PowerJ**

A Sybase product for developing Java applications.

**predicate**

A conditional expression that is optionally combined with the logical operators AND and OR to make up the set of conditions in a WHERE or HAVING clause. In SQL, a predicate that evaluates to UNKNOWN is interpreted as FALSE.
primary key

A column or list of columns whose values uniquely identify every row in the table.

See also: “foreign key” on page 1098.

primary key constraint

A uniqueness constraint on the primary key columns. A table can have only one primary key constraint.

See also:

- “constraint” on page 1093
- “check constraint” on page 1090
- “foreign key constraint” on page 1098
- “unique constraint” on page 1116
- “integrity” on page 1101

primary table

The table containing the primary key in a foreign key relationship.

proxy table

A local table containing metadata used to access a table on a remote database server as if it were a local table.

See also: “metadata” on page 1104.

publication

In MobiLink or SQL Remote, a database object that identifies data that is to be synchronized. In MobiLink, publications exist only on the clients. A publication consists of articles. SQL Remote users can receive a publication by subscribing to it. MobiLink users can synchronize a publication by creating a synchronization subscription to it.

See also:

- “replication” on page 1111
- “article” on page 1089
- “publication update” on page 1108

publication update

In SQL Remote replication, a list of changes made to one or more publications in one database. A publication update is sent periodically as part of a replication message to the remote database(s).

See also:

- “replication” on page 1111
- “publication” on page 1108
**publisher**

In SQL Remote replication, the single user in a database who can exchange replication messages with other replicating databases.

See also: “replication” on page 1111.

**push notification**

In QAnywhere, a special message delivered from the server to a QAnywhere client that prompts the client to initiate a message transmission. In MobiLink server-initiated synchronization, a special message delivered from a Notifier to a device that contains push request data and internal information.

See also:

- “QAnywhere” on page 1109
- “server-initiated synchronization” on page 1112

**push request**

In MobiLink server-initiated synchronization, a row of values in a result set that a Notifier checks to determine if push notifications need to be sent to a device.

See also: “server-initiated synchronization” on page 1112.

**QAnywhere**

Application-to-application messaging, including mobile device to mobile device and mobile device to and from the enterprise, that permits communication between custom programs running on mobile or wireless devices and a centrally located server application.

**QAnywhere agent**

In QAnywhere, a process running on the client device that monitors the client message store and determines when message transmission should occur.

**query**

A SQL statement or group of SQL statements that access and/or manipulate data in a database.

See also: “SQL” on page 1113.

**Redirector**

A web server plug-in that routes requests and responses between a client and the MobiLink server. This plug-in also implements load-balancing and failover mechanisms.

**reference database**

In MobiLink, a SQL Anywhere database used in the development of UltraLite clients. You can use a single SQL Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.
referencing object

An object, such as a view, whose definition directly references another object in the database, such as a table.

See also: “foreign key” on page 1098.

referenced object

An object, such as a table, that is directly referenced in the definition of another object, such as a view.

See also: “primary key” on page 1108.

referential integrity

Adherence to rules governing data consistency, specifically the relationships between the primary and foreign key values in different tables. To have referential integrity, the values in each foreign key must correspond to the primary key values of a row in the referenced table.

See also:

- “primary key” on page 1108
- “foreign key” on page 1098

regular expression

A regular expression is a sequence of characters, wildcards, and operators that defines a pattern to search for within a string.

relational database management system (RDBMS)

A type of database management system that stores data in the form of related tables.

See also: “database management system (DBMS)” on page 1095.

remote database

In MobiLink or SQL Remote, a database that exchanges data with a consolidated database. Remote databases may share all or some of the data in the consolidated database.

See also:

- “synchronization” on page 1115
- “consolidated database” on page 1092

REMOTE DBA authority

In SQL Remote, a level of permission required by the Message Agent (dbremote). In MobiLink, a level of permission required by the SQL Anywhere synchronization client (dbmlsync). When the Message Agent (dbremote) or synchronization client connects as a user who has this authority, it has full DBA access. The user ID has no additional permissions when not connected through the Message Agent (dbremote) or synchronization client (dbmlsync).

See also: “DBA authority” on page 1095.
remote ID

A unique identifier in SQL Anywhere and UltraLite databases that is used by MobiLink. The remote ID is initially set to NULL and is set to a GUID during a database's first synchronization.

replication

The sharing of data among physically distinct databases. Sybase has three replication technologies: MobiLink, SQL Remote, and Replication Server.

Replication Agent

See: “LTM” on page 1103.

replication frequency

In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user.

See also: “replication” on page 1111.

replication message

In SQL Remote or Replication Server, a communication sent between a publishing database and a subscribing database. Messages contain data, passthrough statements, and information required by the replication system.

See also:

● “replication” on page 1111
● “publication update” on page 1108

Replication Server

A Sybase connection-based replication technology that works with SQL Anywhere and Adaptive Server Enterprise. It is intended for near-real time replication between a few databases.

See also: “LTM” on page 1103.

role

In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."

role name

The name of a foreign key. This is called a role name because it names the relationship between the foreign table and primary table. By default, the role name is the table name, unless another foreign key is already using that name, in which case the default role name is the table name followed by a three-digit unique number. You can also create the role name yourself.

See also: “foreign key” on page 1098.
**rollback log**

A record of the changes made during each uncommitted transaction. In the event of a ROLLBACK request or a system failure, uncommitted transactions are reversed out of the database, returning the database to its former state. Each transaction has a separate rollback log, which is deleted when the transaction is complete.

See also: “transaction” on page 1115.

**row-level trigger**

A trigger that executes once for each row that is changed.

See also:

- “trigger” on page 1116
- “statement-level trigger” on page 1114

**schema**

The structure of a database, including tables, columns, and indexes, and the relationships between them.

**script**

In MobiLink, code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.

See also: “event model” on page 1097.

**script-based upload**

In MobiLink, a way to customize the upload process as an alternative to using the log file.

**script version**

In MobiLink, a set of synchronization scripts that are applied together to create a synchronization.

**secured feature**

A feature specified by the -sf option when a database server is started, so it is not available for any database running on that database server.

**server-initiated synchronization**

A way to initiate MobiLink synchronization from the MobiLink server.

**server management request**

A QAnywhere message that is formatted as XML and sent to the QAnywhere system queue as a way to administer the server message store or monitor QAnywhere applications.
server message store

In QAnywhere, a relational database on the server that temporarily stores messages until they are transmitted to a client message store or JMS system. Messages are exchanged between clients via the server message store.

service

In Windows operating systems, a way of running applications when the user ID running the application is not logged on.

session-based synchronization

A type of synchronization where synchronization results in consistent data representation across both the consolidated and remote databases. MobiLink is session-based.

snapshot isolation

A type of isolation level that returns a committed version of the data for transactions that issue read requests. SQL Anywhere provides three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot. When using snapshot isolation, read operations do not block write operations.

See also: “isolation level” on page 1101.

SQL

The language used to communicate with relational databases. ANSI has defined standards for SQL, the latest of which is SQL-2003. SQL stands, unofficially, for Structured Query Language.

SQL Anywhere

The relational database server component of SQL Anywhere that is intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. SQL Anywhere is also the name of the package that contains the SQL Anywhere RDBMS, the UltraLite RDBMS, MobiLink synchronization software, and other components.

SQL-based synchronization

In MobiLink, a way to synchronize table data to MobiLink-supported consolidated databases using MobiLink events. For SQL-based synchronization, you can use SQL directly or you can return SQL using the MobiLink server APIs for Java and .NET.

SQL Remote

A message-based data replication technology for two-way replication between consolidated and remote databases. The consolidated and remote databases must be SQL Anywhere.

SQL statement

A string containing SQL keywords designed for passing instructions to a DBMS.
statement-level trigger

A trigger that executes after the entire triggering statement is completed.

See also:

● “trigger” on page 1116
● “row-level trigger” on page 1112

stored procedure

A stored procedure is a group of SQL instructions stored in the database and used to execute a set of operations or queries on a database server.

string literal

A string literal is a sequence of characters enclosed in single quotes.

subquery

A SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

There are two types of subquery: correlated and nested.

subscription

In MobiLink synchronization, a link in a client database between a publication and a MobiLink user, allowing the data described by the publication to be synchronized.

In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database.

See also:

● “publication” on page 1108
● “MobiLink user” on page 1105

Sybase Central

A database management tool that provides SQL Anywhere database settings, properties, and utilities in a graphical user interface. Sybase Central can also be used for managing other Sybase products, including MobiLink.
synchronization

The process of replicating data between databases using MobiLink technology.

In SQL Remote, synchronization is used exclusively to denote the process of initializing a remote database with an initial set of data.

See also:

- “MobiLink” on page 1105
- “SQL Remote” on page 1113

SYS

A special user that owns most of the system objects. You cannot log in as SYS.

system object

Database objects owned by SYS or dbo.

system table

A table, owned by SYS or dbo, that holds metadata. System tables, also known as data dictionary tables, are created and maintained by the database server.

system view

A type of view, included in every database, that presents the information held in the system tables in an easily understood format.

temporary table

A table that is created for the temporary storage of data. There are two types: global and local.

See also:

- “local temporary table” on page 1102
- “global temporary table” on page 1099

transaction

A sequence of SQL statements that comprise a logical unit of work. A transaction is processed in its entirety or not at all. SQL Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions end either with a COMMIT statement, which makes the changes to the data permanent, or a ROLLBACK statement, which undoes all the changes made during the transaction.

transaction log

A file storing all changes made to a database, in the order in which they are made. It improves performance and allows data recovery in the event the database file is damaged.
transaction log mirror

An optional identical copy of the transaction log file, maintained simultaneously. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.

A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.

See also: “transaction log” on page 1115.

transactional integrity

In MobiLink, the guaranteed maintenance of transactions across the synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.

transmission rule

In QAnywhere, logic that determines when message transmission is to occur, which messages to transmit, and when messages should be deleted.

trigger

A special form of stored procedure that is executed automatically when a user runs a query that modifies the data.

See also:

- “row-level trigger” on page 1112
- “statement-level trigger” on page 1114
- “integrity” on page 1101

UltraLite

A database optimized for small, mobile, and embedded devices. Intended platforms include cell phones, pagers, and personal organizers.

UltraLite runtime

An in-process relational database management system that includes a built-in MobiLink synchronization client. The UltraLite runtime is included in the libraries used by each of the UltraLite programming interfaces, and in the UltraLite engine.

unique constraint

A restriction on a column or set of columns requiring that all non-null values are different. A table can have multiple unique constraints.

See also:

- “foreign key constraint” on page 1098
- “primary key constraint” on page 1108
- “constraint” on page 1093
unload

Unloading a database exports the structure and/or data of the database to text files (SQL command files for the structure, and ASCII comma-separated files for the data). You unload a database with the Unload utility.

In addition, you can unload selected portions of your data using the UNLOAD statement.

upload

The stage in synchronization where data is transferred from a remote database to a consolidated database.

user-defined data type

See “domain” on page 1096.

validate

To test for particular types of file corruption of a database, table, or index.

view

A SELECT statement that is stored in the database as an object. It allows users to see a subset of rows or columns from one or more tables. Each time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views are useful for security purposes, and to tailor the appearance of database information to make data access straightforward.

window

The group of rows over which an analytic function is performed. A window may contain one, many, or all rows of data that has been partitioned according to the grouping specifications provided in the window definition. The window moves to include the number or range of rows needed to perform the calculations for the current row in the input. The main benefit of the window construct is that it allows additional opportunities for grouping and analysis of results, without having to perform additional queries.

Windows

The Microsoft Windows family of operating systems, such as Windows Vista, Windows XP, and Windows 200x.

Windows CE

See “Windows Mobile” on page 1117.

Windows Mobile

A family of operating systems produced by Microsoft for mobile devices.

work table

An internal storage area for interim results during query optimization.
Index

Symbols
-d option
   SQL preprocessor, 590
-e option
   SQL preprocessor, 590
-gn option
   threads, 102
-h option
   SQL preprocessor, 590
-k option
   SQL preprocessor, 590
-m option
   SQL preprocessor, 590
-n option
   SQL preprocessor, 590
-o option
   SQL preprocessor, 590
-q option
   SQL preprocessor, 590
-r option
   SQL preprocessor, 590
-s option
   SQL preprocessor, 590
-u option
   SQL preprocessor, 590
-w option
   SQL preprocessor, 590
-x option
   SQL preprocessor, 590
-z option
   SQL preprocessor, 590

.NET
   (see also ADO.NET)
   data control, 168
   deploying, 1029
   using the SQL Anywhere .NET Data Provider, 109

.NET API
   about, 109

.NET Data Provider
   about, 109
   accessing data, 117
   adding a reference to the DLL in a C# project, 112
   adding a reference to the DLL in a Visual Basic project, 112
   connecting to a database, 114
   connection pooling, 115
   deleting data, 117
   deploying, 139
   error handling, 138
   executing stored procedures, 134
   features, 110
   files required for deployment, 139
   inserting data, 117
   obtaining time values, 132
   POOLING option, 115
   referencing the provider classes in your source code, 112
   registering, 139
   running the sample projects, 111
   supported languages, 4
   system requirements, 139
   tracing support, 141
   transaction processing, 136
   updating data, 117
   using the Simple code sample, 147
   using the Table Viewer code sample, 150
   versions supported, 109

.NET database programming interfaces
   tutorial, 167

.esRepository600
   deploying administration tools on Linux/Unix/Mac OS X, 1068
   deploying administration tools on Windows, 1056

A
   a_backup_db structure
   a_chkpt_log_type, 1007
      syntax, 967
   a_change_log structure
      syntax, 969
   a_chkpt_log_type
      syntax, 1007
   a_create_db structure
      syntax, 971
   a_db_info structure
      syntax, 973
   a_db_version_info structure
      syntax, 975
   a_dblic_info structure
      syntax, 976
   a_dbtools_info structure
Add(Int32, Int32) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 199
Add(Int32, String) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 200
Add(Object) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 401
Add(SABulkCopyColumnMapping) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 198
Add(SqlParameter) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 402
Add(String, Int32) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 201
Add(String, Object) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 403
Add(String, SADbType) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 403
Add(String, SADbType, Int32) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 404
Add(String, SADbType, Int32, String) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 405
Add(String, String) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 201

administration tools
dbtools, 947

ADO
about, 449
Command object, 450
commands, 450
Connection object, 449
connections, 449
cursor types, 39
cursors, 55
introduction to programming, 5
queries, 451
Recordset object, 451, 452
transactions, 454
updates, 453
updating data through a cursor, 453
using SQL statements in applications, 24

ADO.NET
  about, 4
  autocommit mode, 60
  controlling autocommit behavior, 60
  cursor support, 55
  deploying, 1029
  prepared statements, 27
  SQL Anywhere Explorer, 17
  using SQL statements in applications, 24

ADO.NET API
  about, 109
agent IDs
  glossary definition, 1089
alignment of data
  ODBC, 489
All property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 243
alloc_sqlda function
  about, 593
alloc_sqlda_noind function
  about, 593
allusersprofile
  deploying administration tools on Windows, 1056
ALTER EXTERNAL ENVIRONMENT statement
  using, 91
an_erase_db structure
  syntax, 978
an_extfn_api
  external function call API, 674
an_extfn_result_set_column_data
  external function call API, 679
an_extfn_result_set_column_info
  external function call API, 678
an_extfn_result_set_info
  external function call API, 677
an_extfn_value
  external function call API, 676
an_unload_db structure
  syntax, 1000
an_upgrade_db structure
  syntax, 1004
Apache
  choosing a PHP module, 741
  installing, 740

apache_files.txt
  deploying on Mac OS X, 1063
  deploying on Unix, 1060
  deploying on Windows, 1051
apache_license_1.1.txt
  deploying on Mac OS X, 1063
  deploying on Unix, 1060
  deploying on Windows, 1051
apache_license_2.0.txt
  deploying on Mac OS X, 1063
  deploying on Unix, 1060
  deploying on Windows, 1051

APIs
  ADO API, 5
  ADO.NET, 4
  C API, 9
  data access APIs, 3
  JDBC API, 7
  ODBC API, 6
  OLE DB API, 5
  Perl DBD::SQLAnywhere API, 10
  PHP, 754
  Python Database API, 11
  Ruby APIs, 13
  SQL Anywhere C API, 617
  Sybase Open Client API, 15

AppInfo property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 275

application programming interfaces
  (see also APIs)
applications
  deploying, 1017
  deploying .NET clients, 1029
  deploying client applications, 1029
  deploying embedded SQL, 1044
  deploying JDBC clients, 1046
  deploying ODBC, 1037
  deploying OLE DB, 1030
  deploying Open Client, 1046
  SQL, 24

ARRAY clause
  using the FETCH statement, 577
array fetches
  ESQL, 577
articles
  glossary definition, 1089
asensitive cursors
  about, 48
delete example, 42
introduction, 42
update example, 44
ASP.NET
adding the provider schema to the database, 157
registering the connection string, 158
registering the SQL Anywhere ASP.NET providers, 159
using the SQL Anywhere ASP.NET providers, 155
ASP.NET providers
about, 155
atomic transactions
glossary definition, 1089
autocommit
controlling, 60
implementation, 62
JDBC, 516
ODBC, 474
setting for transactions, 60
autoCommit option
configurable option, 1071
AUTOINCREMENT
finding most recent row inserted, 36
autoRefetch
configurable option, 1071
AutoStart property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 275
AutoStop property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 275

B
background processing
callback functions, 589
backups
DBBackup DBTools function, 957
DBTools example, 954
embedded SQL functions, 589
base tables
glossary definition, 1089
BatchSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 182
BeginExecuteNonQuery methods [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 215
BeginExecuteNonQuery() method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 215
BeginExecuteNonQuery(AsyncCallback, Object) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 215
BeginExecuteNonQuery(AsyncCallback, Object, CommandBehavior) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 219
BeginExecuteNonQuery(CommandBehavior) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 217
BeginTransaction methods [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 254
BeginTransaction() method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 254
BeginTransaction(IsolationLevel) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 255
BeginTransaction(SAIsolationLevel) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 256
benefits of using cursors
about, 30
BIGINT data type
embedded SQL, 547
BINARY data type
embedded SQL, 547
bind parameters
prepared statements, 27
bind variables
about, 561
bit arrays
glossary definition, 1090
BIT data type
embedded SQL, 547
bit fields
using, 953
blank padding
strings in embedded SQL, 542
blank padding enumeration
syntax, 1007
BLOBs
embedded SQL, 582
retrieving in embedded SQL, 583
sending in embedded SQL, 584
CALL statement
  embedded SQL, 586
callback functions
  embedded SQL, 589
  registering, 604
callbacks
  DB_CALLBACK_CONN_DROPPED, 605
  DB_CALLBACK_DEBUG_MESSAGE, 604
  DB_CALLBACK_FINISH, 605
  DB_CALLBACK_MESSAGE, 605
  DB_CALLBACK_START, 605
  DB_CALLBACK_VALIDATE_FILE_TRANSFER, 606
  DB_CALLBACK_WAIT, 605
calling external libraries from procedures
  about, 670
Cancel method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 220
cancel processing
  in external functions, 683
canceling requests
  embedded SQL, 589
CanCreateDataSourceEnumerator property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 362
capabilities
  supported, 847
carriers
  glossary definition, 1090
case sensitivity
  Java in the database and SQL, 86
CD-ROM
  deploying databases on, 1078
chained mode
  controlling, 60
  implementation, 62
  transactions, 60
chained option
  JDBC, 516
ChangeDatabase method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 256
ChangePassword method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 257
character data
  character sets in Embedded SQL, 549
  length in Embedded SQL, 550
character sets
  glossary definition, 1090
Index

HTTP requests, 941
setting CHAR character set, 598
setting NCHAR character set, 598
character strings
  embedded SQL, 590
CharArray property [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 276
CHECK constraints
  glossary definition, 1090
checkForUpdates
  configurable option, 1071
checkpoints
  glossary definition, 1090
checksums
  glossary definition, 1091
cis.zip
  deploying database servers, 1074
Class.forName method
  loading iAnywhere JDBC driver, 505
  loading jConnect, 508
classes
  creating, 98
  installing, 98
  runtime, 85
  updating, 100
  versions, 100
CLASSPATH environment variable
  Java in the database, 94
  jConnect, 507
  setting, 513
clauses
  WITH HOLD, 34
Clear method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 408
ClearAllPools method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 258
ClearPool method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 258
client
  time change, 611
client message store IDs
  glossary definition, 1091
client message stores
  glossary definition, 1091
client side autocommit
  about, 62
Client-Library
  Sybase Open Client, 840
classpath
  glossary definition, 1091
ClientPort property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 434
close method
  Python, 735
Close method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 185, 258, 317
CLOSE statement
  using cursors in embedded SQL, 575
CLR keyword
  external environment, 694
code pages
  glossary definition, 1091
CodeXchange
  samples, 854
collations
  glossary definition, 1091
ColumnMappings property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 183
Columns field [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 374
Command ADO object
  ADO, 450
command files
  glossary definition, 1091
command line utilities
  deploying, 1082
command prompts
  conventions, xv
  curly braces, xv
  environment variables, xv
  parentheses, xv
  quotes, xv
Command property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 425, 428
command shells
  conventions, xv
  curly braces, xv
  environment variables, xv
  parentheses, xv
  quotes, xv
commands
  ADO Command object, 450
CommandText property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 211
CommandTimeout property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 211
CommandType property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 212
CommBufferSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 276
commit method
Python, 736
Commit method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 443
COMMIT statement
cursors, 62
JDBC, 516
commOnExit
configurable option, 1071
committing
transactions from ODBC, 474
CommitTrans ADO method
ADO programming, 454
updating data, 454
CommLinks property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 276
communication streams
glossary definition, 1092
compile and link process
about, 532
compilers
supported, 533
components
transaction attribute, 72
Compress property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 277
CompressionThreshold property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 277
concurrency
glossary definition, 1092
concurrency values
SQL_CONCUR_LOCK, 492
SQL_CONCUR_READ_ONLY, 492
SQL_CONCUR_ROWVER, 492
SQL_CONCUR_VALUES, 492
configuring
administration tools for deployment, 1071
Interactive SQL for deployment, 1071
Sybase Central for deployment, 1071
configuring the SQL Anywhere Explorer
about, 20
conflict resolution
glossary definition, 1092
conformance
ODBC, 464
connect method
Python, 735
Connection ADO object
ADO, 449
ADO programming, 454
connection handles
ODBC, 473
connection IDs
glossary definition, 1092
connection pooling
.NET Data Provider, 115
connection profiles
glossary definition, 1092
Connection property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 212, 441
connection state
.NET Data Provider, 115
connection-initiated synchronization
glossary definition, 1092
CONNECTION_PROPERTY function
element, 940
ConnectionLifetime property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 278
ConnectionName property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 278
ConnectionReset property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 278
connections
ADO Connection object, 449
connecting to a database using the .NET Data Provider, 114
functions, 610
iAnywhere JDBC driver URL, 505
jConnect, 509
jConnect URL, 509
JDBC, 504
JDBC client applications, 511
JDBC defaults, 516
JDBC example, 511, 514
JDBC in the server, 514
licensing web applications, 938
ODBC functions, 476
ODBC getting attributes, 478
ODBC programming, 476
not supported with Open Client, 847
CS_DATA_SENSITIVITY
not supported with Open Client, 847
CS_PROTO_DYNPROC
not supported with Open Client, 847
CS_REG_NOTIF
not supported with Open Client, 847
CS_REQ_BCP
not supported with Open Client, 847
ct_command function
  Open Client, 844, 846
t_ct_cursor function
  Open Client, 844
t_ct_dynamic function
  Open Client, 844
t_ct_results function
  Open Client, 846
t_ct_send function
  Open Client, 846
cursor positioning
troubleshooting, 34
cursor positions
glossary definition, 1093
cursor result sets
glossary definition, 1093
cursor sensitivity and performance
about, 50
cursors
about, 29
ADO, 55
ADO.NET, 55
asensitive, 48
availability, 39
benefits, 30
block cursors, 40
canceling, 38
choosing ODBC cursor characteristics, 492
db_cancel_request function, 597
delete, 845
describing result sets, 58
dynamic, 46
DYNAMIC SCROLL and asensitive cursors, 48
DYNAMIC SCROLL and cursor positioning, 34
embedded SQL supported types, 56
embedded SQL usage, 575
example C code, 538
fat, 35
fetching multiple rows, 35
fetching rows, 33, 34
glossary definition, 1093
insensitive, 39, 46
inserting multiple rows, 36
inserting rows, 36
internals, 41
introduction, 29
isolation level, 34
keyset-driven, 49
membership, 41
NO SCROLL, 39
ODBC, 55, 491
ODBC bookmarks, 494
ODBC deletes, 494
ODBC result sets, 493
ODBC updates, 494
OLE DB, 55
Open Client, 844
order, 41
performance, 50, 51
platforms, 39
positioning, 33
prepared statements, 32
properties, 39
Python, 736
read-only, 39, 46
requesting, 55
result sets, 29
savepoints, 63
SCROLL, 39, 49
scrollable, 35
sensitive, 46
sensitivity, 41, 42
sensitivity and isolation levels, 54
sensitivity examples, 42, 44
static, 46
step-by-step, 32
stored procedures, 586
transactions, 62
unique, 39
unspecified sensitivity, 48
update, 845
updating, 453
updating and deleting rows, 36
uses, 29
using, 32
value-sensitive, 49
values, 41
visible changes, 41
work tables, 50
cursors and bookmarks
  about, 40

D
data
  accessing with the .NET Data Provider, 117
  manipulating with the .NET Data Provider, 117
data connection
  Visual Studio, 168
data cube
glossary definition, 1094
data manipulation language
glossary definition, 1094
data type conversion
  indicator variables, 553
data types
  C data types, 547
dynamic SQL, 565
eMBEDDED SQL, 542
glossary definition, 1094
host variables, 547
in web services handlers, 880
mapping, 842
Open Client, 842
ranges, 842
SQL and C, 685
SQLDA, 566
DataAdapter
  about, 117
deleting data, 124
inserting data, 124
obtaining primary key values, 129
obtaining result set schema information, 128
retrieving data, 123
updating data, 124
using, 123
DataAdapter property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 232
database administrator
glossary definition, 1094
database connections
glossary definition, 1095
database files
glossary definition, 1095
database management
data types
  C data types, 547
dynamic SQL, 565
eMBEDDED SQL, 542
glossary definition, 1094
host variables, 547
in web services handlers, 880
mapping, 842
Open Client, 842
ranges, 842
SQL and C, 685
SQLDA, 566
iAnywhere.Data.SQLAnywhere namespace, 252
database server
  deploying, 1074
  functions, 610
  glossary definition, 1095
database size enumeration
  syntax, 1008
database tools interface
  a_backup_db structure, 967
  a_change_log structure, 969
  a_chkpt_log_type enumeration, 1007
  a_create_db structure, 971
  a_db_info structure, 973
  a_db_version_info structure, 975
  a_dblic_info structure, 976
  a_dbtools_info structure, 977
  a_name structure, 978
  a_remote_sql structure, 979
  a_sync_db structure, 985
  a_syncpub structure, 993
  a_sysinfo structure, 993
  a_table_info structure, 994
  a_translate_log structure, 995
  a_truncate_log structure, 999
  a_validate_db structure, 1005
  a_validate_type enumeration, 1010
  about, 947
  an_erase_db structure, 978
  an_unload_db structure, 1000
  an_upgrade_db structure, 1004
  Blank padding enumeration, 1007
database size enumeration, 1008
Database version enumeration, 1008
DBBackup function, 957
DBChangeLogName function, 957
DBCCreate function, 958
DB_CALLBACK_WAIT callback parameter
about, 605

DB_CAN_MULTI_CONNECT
    db_find_engine function, 599

DB_CAN_MULTI_DB_NAME
    db_find_engine function, 599

db_cancel_request function
    about, 597
    request management, 589

db_change_char_charset function
    about, 598

db_change_nchar_charset function
    about, 598

DB_CLIENT
    db_find_engine function, 599

DB_CONNECTION_DIRTY
    db_find_engine function, 599

DB_DATABASE_SPECIFIED
    db_find_engine function, 599

db_delete_file function
    about, 599

DB_ENGINE
    db_find_engine function, 599

db_find_engine function
    about, 599

db_fini function
    about, 600

db_fini_dll
    calling, 536

db_get_property function
    about, 600

db_init function
    about, 601

db_init_dll
    calling, 536

db_is_working function
    about, 601
    request management, 589

db_locate_servers function
    about, 602

db_locate_servers_ex function
    about, 603

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
    about, 603

DB_LOOKUP_FLAG_DATABASES
    db_find_engine function, 599

DB_LOOKUP_FLAG_NUMERIC
    about, 603

DB_NO_DATABASES
    db_find_engine function, 599

DB_PROP_CLIENT_CHARSET
    usage, 601

DB_PROP_DBLIB_VERSION
    usage, 601

DB_PROP_SERVER_ADDRESS
    usage, 601

db_register_a_callback function
    about, 604
    request management, 589

db_start_database function
    about, 607

db_start_engine function
    about, 607

db_stop_database function
    about, 608

db_stop_engine function
    about, 609

db_string_connect function
    about, 610

db_string_disconnect function
    about, 610

db_string_ping_server function
    about, 611

db_time_change function
    about, 611

DBA authority
    glossary definition, 1095

DBBackup function
    about, 957

dbcapi.dll
    deploying, 1079
    PHP support module, 716

DBChangeLogName function
    about, 957

dbcis11.dll
    deploying database servers, 1074
    deploying database utilities, 1082
    deploying embedded SQL clients, 1045
    deploying ODBC clients, 1037
    deploying OLE DB clients, 1030
    deploying on Windows, 1051

dbconsole utility

1130 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1
deploying, 1048
deploying on Linux and Unix, 1059
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows without InstallShield, 1049
dbconsole.exe
  deploying on Windows, 1051
dbconsole.ini
  deploying administration tools, 1048
DBConsole.jar
  deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
DBCreate function
  about, 958
DBCreatedVersion function
  about, 958
dbctrs11.dll
  deploying database servers, 1074
deploying SQL Anywhere, 1077
DBD::SQLAnywhere
  about, 721
deploying on Unix and Mac OS X, 725
deploying on Windows, 723
writing Perl scripts, 727
dbecc11.dll
  deploying embedded SQL clients, 1045
  ECC encryption, 1081
dbelevate11.exe
  deploying on Windows, 1051
  registering DLLs, 1077
dbencod11.dll
  deploying SQL Remote, 1084
dbeng11
  deploying SQL Remote, 1084
dbeng11.exe
  deploying database servers, 1074
dbeng11.lic
  deploying database servers, 1074
dbeng11.dll
  deploying database servers, 1074
DBErase function
  about, 959
dbextclr11.exe
  deploying, 1079
dbextenv11.dll
  deploying, 1079
PHP support module, 716
dbexternc11
  deploying, 1079
dbextmc11.exe
  deploying, 1079
  deploying database servers, 1074
dbf1e11.dll
  deploying SQL Remote, 1084
dbfips11.dll
  deploying embedded SQL clients, 1045
  FIPS-approved AES encryption, 1081
  FIPS-approved RSA encryption, 1081
dbfzp11.dll
  deploying SQL Remote, 1084
dbghelp.dll
  deploying database servers, 1074
  deploying on Windows, 1051
DBI module (see DBD::SQLAnywhere)
dbicu11.dll
  deploying database servers, 1074
  deploying ODBC clients, 1037
  deploying on Windows, 1051
dbicudt11.dat
  deploying database servers, 1074
  deploying ODBC clients, 1037
dbicudt11.dll
  deploying database servers, 1074
  deploying ODBC clients, 1037
  deploying on Windows, 1051
DBInfo function
  about, 959
DBInfoDump function
  about, 960
DBInfoFree function
  about, 960
dbinit utility
  deploying on Mac OS X, 1063
deploying on Unix, 1060
deployment considerations, 1083
dbinit.exe
  deploying on Windows, 1051
dbisql
  deploying on Mac OS X, 1063
deploying on Unix, 1060
dbisql.com
  deploying on Windows, 1051
dbisql.exe
  deploying on Windows, 1051
dbisql.ini
deploying administration tools, 1048

dbisqlc utility
  deploying, 1072
  limited functionality, 1072
  Unix supported deployment platforms, 1059
dbjodbc11.dll
  deploying, 1079
  deploying database servers, 1074
  deploying JDBC clients, 1046
  deploying on Windows, 1051
dblgen11.dll
  deploying database servers, 1074
  deploying database utilities, 1082
  deploying embedded SQL clients, 1045
  deploying ODBC clients, 1037
  deploying OLE DB clients, 1030
  deploying on Windows, 1051
  deploying SQL Remote, 1084
dblgen11.dll registry entry
  about, 1075
dblgen11.res
  deploying database servers, 1074
  deploying database utilities, 1082
  deploying ODBC clients, 1037
  deploying on Mac OS X, 1063
  deploying on Unix, 1060
  deploying SQL Remote, 1084

DBLIB
  dynamic loading, 536
  interface library, 532
dlib11.dll
  deploying database utilities, 1082
  deploying embedded SQL clients, 1045
  deploying on Windows, 1051
  deploying SQL Remote, 1084

DBLicense function
  about, 961
dbmlsync utility
  building your own, 985
  C API for, 985
dbmlsynccom.dll
  deploying SQL Anywhere, 1077
dbmlsynccomg.dll
  deploying SQL Anywhere, 1077

DBMS
  glossary definition, 1095
dbodbc11.bundle
  deploying ODBC clients, 1037
     Mac OS X ODBC driver, 469
dbodbc11.dll
  deploying database servers, 1074
  deploying ODBC clients, 1037
  deploying on Windows, 1051
  deploying SQL Anywhere, 1077
  linking, 466
dbodbc11.lib
  Windows Mobile ODBC import library, 467
dbodbc11_r.bundle
  deploying ODBC clients, 1037
dboftsp.dll
  deploying database utilities, 1082
dboledb11.dll
  deploying OLE DB clients, 1030
  deploying SQL Anywhere, 1077
dboledba11.dll
  deploying OLE DB clients, 1030
  deploying SQL Anywhere, 1077
dbput11.dll
  deploying on Windows, 1051
dbremote
  deploying SQL Remote, 1084

DBRemoteSQL function
  about, 961
dbrmt.h
  about, 948
  database tools interface, 967
dbrsa11.dll
  RSA encryption, 1081
dbrsakp11.dll
  deploying database servers, 1074
  deploying JDBC clients, 1046
  deploying Open Client, 1046
dbscript11.dll
  deploying database servers, 1074
dbserv11.dll
  AES encryption, 1081
  deploying database servers, 1074
dbsmtp11.dll
  deploying SQL Remote, 1084
dbspaces
  glossary definition, 1096
dbrsr11
  deploying database servers, 1074
dbsrv11.exe
  deploying database servers, 1074
dbsrv11.lic
DDL glossary definition, 1094
deadlocks glossary definition, 1096
debugger.jar deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
DECIMAL data type embedded SQL, 547
DECL_BIGINT macro about, 547
DECL_BINARY macro about, 547
DECL_BIT macro about, 547
DECL_DATE macro about, 547
DECL_DATETIME macro about, 547
DECL_DECIMAL macro about, 547
DECL_FIXCHAR macro about, 547
DECL_LONG_BINARY macro about, 547
DECL_LONGNVARCHAR macro about, 547
DECL_LONGVARCHAR macro about, 547
DECL_NCHAR macro about, 547
DECL_NFIXCHAR macro about, 547
DECL_NVARCHAR macro about, 547
DECL_UNSIGNED_BIGINT macro about, 547
DECLVARCHAR macro about, 547
DECLARE statement declaration section about, 546
DECLARE statement using cursors in embedded SQL, 575
declaring embedded SQL data types, 542
host variables, 546
DELETE statement JDBC, 517
positioned, 36
DeleteCommand property [SA .NET 2.0] iAnywhere.Data.SQLAnywhere namespace, 304
DeleteDynamic method JDBCExample, 520
DeleteStatic method JDBCExample, 518
deploying .NET data provider, 1029
.NET Data Provider applications, 139
about, 1017
administration tools, 1048
administration tools on Linux and Unix, 1059
administration tools on Windows without InstallShield, 1049
ADO.NET data provider, 1029
applications and databases, 1017
client applications, 1029
console utility [dbconsole], 1048
console utility [dbconsole] on Linux and Unix, 1059
console utility [dbconsole] on Windows without InstallShield, 1049
database servers, 1074
databases, 1077
databases on CD-ROM, 1078
deployment wizard, 1023
embedded databases, 1082
embedded SQL, 1044
file locations, 1020
iAnywhere JDBC driver, 1046
Interactive SQL, 1048
Interactive SQL (dbisqlc), 1072
Interactive SQL on Linux and Unix, 1059
Interactive SQL on Windows without InstallShield, 1049
international considerations, 1078
Java in the database, 1074
jConnect, 1046
JDBC clients, 1046
localizing databases, 1078
MobiLink plug-in, 1049
ODBC, 1037
ODBC data sources, 1041
OLE DB provider, 1030
Open Client, 1046
overview, 1018
personal database server, 1082
QAnywhere plug-in, 1049
read-only databases, 1078
registering DLLs, 1077
registry settings, 1075
silent install, 1027
SQL Anywhere, 1017
SQL Anywhere components on Windows, 1023
SQL Anywhere plug-in, 1049
SQL Remote, 1084
Sybase Central, 1048
Sybase Central on Linux and Unix, 1059
Sybase Central on Windows without InstallShield, 1049
UltraLite plug-in, 1049
Unix issues, 1020
wizard, 1023
deploying on Linux and Unix
SQL Anywhere Console [dbconsole] utility, 1059
deploying the SQL Anywhere .NET Data Provider
about, 139
deployment wizard
about, 1023
Depth property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 313
DeriveParameters method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 233
DESCRIBE statement
about, 563
multiple result sets, 588
SQLDA fields, 566
sqllen field, 568
sqltype field, 568
describing
NCHAR columns in Embedded SQL, 568
result sets, 58
descriptors
describing result sets, 58
DesignTimeVisible property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 213
DestinationColumn property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 194
DestinationOrdinal property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 194
DestinationTableName property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 184
developer community
newsgroups, xvii
developing applications with the .NET Data Provider
about, 109
developing applications with the ASP.NET providers
about, 155
device tracking
glossary definition, 1096
direct row handling
glossary definition, 1096
Direction property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 388
directory structure
Unix, 1020
disableExecuteAll
configurable option, 1071
DisableMultiRowFetch property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 281
disableResultsEditing
configurable option, 1071
DISH services
about, 849
creating, 856, 865
JAX-WS tutorial, 871, 892
Microsoft .NET tutorial, 868, 887
Dispose method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 186
Distributed Transaction Coordinator
three-tier computing, 68
distributed transaction processing
using the .NET Data Provider, 137
distributed transactions
about, 66
architecture, 69
EAServer, 72
enlistment, 68
recovery, 71
restrictions, 70
three-tier computing, 67
DLL entry points
about, 593
DLLs
calling from stored procedures, 670
deployment, 1077
external procedure calls, 672
multiple SQLCAs, 559
registering for deployment, 1077
DML
glossary definition, 1094
DoBroadcast property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 434
DocCommentXchange (DCX)
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>about, xii</td>
</tr>
<tr>
<td>documentation</td>
</tr>
<tr>
<td>conventions, xiv</td>
</tr>
<tr>
<td>SQL Anywhere, xii</td>
</tr>
<tr>
<td>domains</td>
</tr>
<tr>
<td>glossary definition, 1096</td>
</tr>
<tr>
<td>downloads</td>
</tr>
<tr>
<td>glossary definition, 1096</td>
</tr>
<tr>
<td>drivers</td>
</tr>
<tr>
<td>deploying the SQL Anywhere ODBC driver, 1037</td>
</tr>
<tr>
<td>iAnywhere JDBC driver, 502</td>
</tr>
<tr>
<td>jConnect JDBC driver, 502</td>
</tr>
<tr>
<td>linking the SQL Anywhere ODBC driver on Windows, 466</td>
</tr>
<tr>
<td>DSN-less connections</td>
</tr>
<tr>
<td>using ODBC, 1044</td>
</tr>
<tr>
<td>DT_BIGINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_BINARY embedded SQL data type</td>
</tr>
<tr>
<td>about, 543</td>
</tr>
<tr>
<td>DT_BIT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_DATE embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_DECIMAL embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_DOUBLE embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_FIXCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_FLOAT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_INT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_LONGBINAR Y embedded SQL data type</td>
</tr>
<tr>
<td>about, 544</td>
</tr>
<tr>
<td>DT_LONGNVARCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 543</td>
</tr>
<tr>
<td>DT_LONGVARCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 543</td>
</tr>
<tr>
<td>DT_NFIXCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_NSTRING embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_NVARCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 543</td>
</tr>
<tr>
<td>DT_SMALLINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_STRING data type</td>
</tr>
<tr>
<td>about, 614</td>
</tr>
<tr>
<td>DT_STRING embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_TIME embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_TIMESTAMP embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_TIMESTAMP_STRUCT embedded SQL data type</td>
</tr>
<tr>
<td>about, 544</td>
</tr>
<tr>
<td>DT_TINYINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 544</td>
</tr>
<tr>
<td>DT_UNSBIGINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_UNSINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_UNSSMALLINT embedded SQL data type</td>
</tr>
<tr>
<td>about, 542</td>
</tr>
<tr>
<td>DT_VARCHAR embedded SQL data type</td>
</tr>
<tr>
<td>about, 543</td>
</tr>
<tr>
<td>DT_VARIABLE embedded SQL data type</td>
</tr>
<tr>
<td>about, 544</td>
</tr>
<tr>
<td>DTC</td>
</tr>
<tr>
<td>isolation levels, 70</td>
</tr>
<tr>
<td>three-tier computing, 68</td>
</tr>
<tr>
<td>DTC isolation levels</td>
</tr>
<tr>
<td>about, 70</td>
</tr>
<tr>
<td>dynamic cursors</td>
</tr>
<tr>
<td>about, 46</td>
</tr>
<tr>
<td>ODBC, 55</td>
</tr>
<tr>
<td>sample, 540</td>
</tr>
<tr>
<td>DYNAMIC SCROLL cursors</td>
</tr>
<tr>
<td>asensitive cursors, 48</td>
</tr>
<tr>
<td>embedded SQL, 56</td>
</tr>
<tr>
<td>troubleshooting, 34</td>
</tr>
<tr>
<td>dynamic SQL</td>
</tr>
<tr>
<td>about, 561</td>
</tr>
<tr>
<td>glossary definition, 1096</td>
</tr>
<tr>
<td>SQLDA, 565</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAserver</td>
</tr>
<tr>
<td>component transaction attribute, 72</td>
</tr>
<tr>
<td>distributed transactions, 72</td>
</tr>
<tr>
<td>three-tier computing, 68</td>
</tr>
<tr>
<td>transaction coordinator, 72</td>
</tr>
</tbody>
</table>
EBFs
glossary definition, 1096

Elevate property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 281

embedded databases
deploying, 1082

embedded SQL
about, 531
authorization, 590
autocommit mode, 60
character strings, 590
compile and link process, 532
controlling autocommit behavior, 60
cursor types, 39
cursors, 56, 538, 575
deploying clients, 1044
development, 532
dynamic cursors, 540
dynamic statements, 561
example program, 535
FETCH FOR UPDATE, 54
fetching data, 574
functions, 593
glossary definition, 1097
header files, 534
host variables, 546
import libraries, 534
introduction to programming, 8
line numbers, 590
SQL statements, 24
statement summary, 615
static statements, 561
encoding
glossary definition, 1097
EncryptedPassword property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 282
Encryption property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 282
EndExecuteNonQuery method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 221
EndExecuteReader method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 223
Enlist property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 283
EnlistDistributedTransaction method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 259
enlistment
distributed transactions, 68

EnlistTransaction method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 260
Enterprise Application Server (see EAServer)
entry points
calling DBTools functions, 951
enumerations
DBTools interface, 1007
environment handles
ODBC, 473
environment variables
command prompts, xv
command shells, xv
for Java in the database, 91
error codes
SQL Anywhere exit codes, 1013
error handling
.NET Data Provider, 138
Java, 84
ODBC, 498
error messages
eMBED SQL function, 614
errors
HTTP codes, 942
SOAP faults, 942
sqlcode SQLCA field, 555
Errors property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 357, 367
escape characters
Java in the database, 87
SQL, 87
escape syntax
Interactive SQL, 526
esqdll.c
about, 536
event log
EventLogMask, 1076
registry entry, 1075
event model
glossary definition, 1097
exceptions
Java, 84
EXEC SQL
embedded SQL development, 536
execute method
Python, 736
EXECUTE statement
about, 561

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1 1137
stored procedures in embedded SQL, 586
execute many method
Python, 737
ExecuteNonQuery method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 225
ExecuteReader method
SACommand class, 149, 153
using, 27, 118
ExecuteReader methods [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 226
ExecuteReader() method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 226
ExecuteReader(CommandBehavior) method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 226
ExecuteScalar method
using, 119
ExecuteScalar method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 227
executeToolbarButtonSemantics
configurable option , 1071
executeUpdate JDBC method
about, 517
using, 28
executing SQL statements
in applications, 24
exit codes
about, 1013
Explorer (see SQL Anywhere Explorer )
external calls
creating procedures and functions, 671
external environments
about, 690
C API, 697
CLR, 694
embedded SQL, 697
JAVA, 706
ODBC, 697
PERL, 711
PHP, 715
external function call API
an_extfn_api, 674
an_extfn_result_set_column_data, 679
an_extfn_result_set_column_info, 678
an_extfn_result_set_info, 677
an_extfn_value, 676
extfn_cancel method, 674
extfn_use_new_api method, 673
get_piece callback, 681
get_value callback, 681
prototype, 673
set_cancel callback, 683
set_value callback, 682
external function calls
about, 669, 689
external functions
canceling, 683
passing parameters, 681
prototypes, 673
return types, 686
unloading libraries, 688
external logins
glossary definition, 1097
external objects
about, 692
comment, 692
system table, 691
external procedure calls
about, 669, 689
extfn_cancel
external function call API, 674
extfn_use_new_api
external function call API, 673
extraction
glossary definition, 1097
F
failover
glossary definition, 1097
fastLauncherEnabled
configurable option , 1071
fat cursors
about, 35
feedback
documentation, xvii
providing, xvii
reporting an error, xvii
requesting an update, xvii
FETCH FOR UPDATE
embedded SQL, 54
ODBC, 54
fetch operation
cursors, 34
multiple rows, 35
scrollable cursors, 35
Index

GetByte method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 318
GetBytes method
  using, 132
GetBytes method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 319
GetChar method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 320
GetChars method
  using, 132
GetChars method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 321
getConnection method
  instances, 516
GetData method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 322
GetDataSources method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 342
GetDataTypeName method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 322
GetDateTime method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 323
GetDecimal method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 324
GetDeleteCommand methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 233
GetDeleteCommand() method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 234
GetDeleteCommand(Boolean) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 233
GetDouble method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 325
GetEnum method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 325, 355, 410
GetEnum method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 326
GetEnum method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 327
GetInsertCommand methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 235
GetInsertCommand() method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 236
GetInsertCommand(Boolean) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 237
GetUpdateCommand methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 238
GetUpdateCommand() method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 239
GetUpdateCommand(Boolean) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 240
GetInt16 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 328
GetInt32 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 328
GetInt64 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 328
GetKeyword method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 295
GetLength method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 330
GetObjectData method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 359
GetOrdinal method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 330
GetSchema methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 260
GetSchema() method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 261
GetSchema(String) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 261
GetSchema(String, String[]) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 262
GetSchemaTable method
  using, 122
GetSchemaTable method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 331
GetString method
  iAnywhere.Data.SQLAnywhere namespace, 333
GetTimeSpan method
  using, 132
GetTimeSpan method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 334
getting help
  technical support, xvii
GetUInt16 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 335
GetUInt32 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 335
GetUInt64 method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 336
GetUpdateCommand methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 237
GetUpdateCommand() method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 238
GetUpdateCommand(Boolean) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 239

HTML web server
  quick start, 851
HTML_DECODE
  example, 877
HTTP
  default services, 878
HTTP headers
  in web services handlers, 919
  modifying, 903
  suppressing, 903
HTTP server
  about, 849
  character sets, 941
  creating web services, 856
  data types, 880
  errors, 942
  HTTP headers, 919
  HTTP session, 935
  interpreting URLs, 861
  quick start, 851
  request handlers, 877
  variables, 917
HTTP services
  listening for SOAP HTTP requests, 859
HTTP session
  about, 935
  connections, 937
  errors, 938
  semantics, 937
  timeout, 938
HTTP_HEADER function
  web services, 919
http_session_timeout option
  web services, 938
HTTP_VARIABLE function
  web services, 917

i
iAnywhere developer community
  newsgroups, xvii
iAnywhere JDBC driver
  choosing a JDBC driver, 502
  connecting, 505
  deploying JDBC clients, 1046
  glossary definition JDBC clients, 1046
  loading, 505
  required files, 505

H
handles
  about ODBC, 473
  allocating ODBC, 473
hash
  glossary definition, 1099
HasRows property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 314
header files
  embedded SQL, 534
  ODBC, 466
help
  technical support, xvii
histograms
  glossary definition, 1099
Host property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 435
host variables
  about, 546
  data types, 547
  declaring, 546
  not supported in batches, 546
  SQLDA, 566
  uses, 550
about, 70
isql.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
IsReadOnly property [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 398
IsSynchronized property [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 399
Item properties [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 315, 399
Item property [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 198, 293, 354
Item(Int32) property [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 315, 399
Item(String) property [SA .NET 2.0]  
iAnywhere.Data.SQLAnywhere namespace, 316, 400

J
Jaguar (see EAServer)
J AR and ZIP file creation wizard
using, 99
J AR files
adding, 99
glossary definition, 1101
installing, 98, 99
updating, 100
versions, 100
Java
JDBC, 501
storing classes, 80
Java API for XML Web Services
about, 871
Java class creation wizard
using, 515
Java classes
adding, 98
glossary definition, 1101
installing, 98
Java in the database
about, 77
API, 85
choosing a Java VM, 91
deploying, 1074
environment variables, 91
error handling, 84
escape characters, 87
installing classes, 98
key features, 80
main method, 87, 102
NoSuchMethodException, 102
persistence, 87
returning result sets, 103
runtime environment, 85
security management, 105
starting the VM, 106
stopping the VM, 106
supported platforms, 81
virtual machine, 80
JAVA keyword
external environment, 706
Java Runtime Environment
using Java in the database, 91
Java signatures
CREATE PROCEDURE statement [user defined], 707
Java stored procedures
about, 103
e xample, 103
Java VM
JAVA_HOME environment variable, 91
JAVAHOME environment variable, 91
selecting, 91
starting, 106
stopping, 106
JAVA_HOME environment variable
starting the Java VM, 91
deprecated, 91
java_location option
java_main_userid option
java_vm_options option
using, 91
JAVAHOME environment variable
starting the Java VM, 91
JAX-WS
about, 871
accessing web services tutorial, 871
using data types tutorial, 892
JComponents1101.jar
deploying on Mac OS X, 1063
32-bit Java Runtime Environment, 1050
deploying on Windows, 1051
jre_1.6.0_linux_sun_i586
deploying on Mac OS X, 1063
deploying on Unix, 1060
jre_1.6.0_solaris_sun_sparc
deploying on Mac OS X, 1063
deploying on Unix, 1060
JSON server
creating web services, 856
JSON web server
quick start, 851
jsybli600.dll
deploying on Windows, 1051
jsybli600.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051

K
keep-alive request-header field
HTTP headers, 919
Kerberos property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 284
key joins
glossary definition, 1099
keyHH.exe
deploying on Windows, 1051
Keys property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 294
keyset-driven cursors
about, 49
ODBC, 55

L
Language property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 285
languages
file names, 1021
LDAP property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 436
length SQLDA field
about, 566, 568
libdbcapi.dylib
PHP support module, 716
libdbcapi.so
PHP support module, 716
libdbcapi_r.dylib
deploying, 1079
PHP support module, 716
libdbcapi_r.so
deploying, 1079
PHP support module, 716
libdbcis11.dylib
deploying database servers, 1074
libdbcis11.so
deploying database servers, 1074
libdbec11.so
ECC encryption, 1081
libdbencod11_r
deploying SQL Remote, 1084
libdbextenv11_r.dylib
deploying database servers, 1074
libdbextenv11_r.so
deploying database servers, 1074
libdbextf.dylib
deploying database servers, 1074
libdbextf.so
deploying database servers, 1074
libdbfile11_r
deploying SQL Remote, 1084
libdbftp11_r
deploying SQL Remote, 1084
libdbicu11
deploying ODBC clients, 1037
libdbicu11_r.dylib
deploying database servers, 1074
libdbicu11_r.so
deploying database servers, 1074
libdbicudt11.dylib
deploying database servers, 1074
libdbicudt11.so
deploying database servers, 1074
libdbicudt11_r.dylib
deploying database servers, 1074
libdbicudt11_r.so
deploying database servers, 1074

Index
deploying on Mac OS X, 1063
libdbjodbc11.dylib
  deploying, 1079
  deploying database servers, 1074
  deploying JDBC clients, 1046
  deploying on Mac OS X, 1063
libdbjodbc11.so
  deploying, 1079
  deploying database servers, 1074
  deploying JDBC clients, 1046
libdbjodbc11.so.1
  deploying on Unix, 1060
libdblib11.dylib
  deploying database utilities, 1082
libdblib11.so
  deploying database utilities, 1082
  deploying embedded SQL clients, 1045
libdblib11_r
  deploying SQL Remote, 1084
libdblib11_r.dylib
  deploying database utilities, 1082
  deploying on Mac OS X, 1063
libdblib11_r.so
  deploying database utilities, 1082
libdblib11_r.so.1
  deploying on Unix, 1060
libdbodbc11
  Unix ODBC driver, 468
libdbodbc11.
  deploying ODBC clients, 1037
libdbodbc11.dylib
  deploying database servers, 1074
libdbodbc11.so
  deploying database servers, 1074
libdbodbc11_n
  deploying ODBC clients, 1037
libdbodbc11_n.dylib
  deploying database servers, 1074
libdbodbc11_n.so
  deploying database servers, 1074
libdbodbc11_r
  deploying ODBC clients, 1037
libdbodbc11_r.dylib
  deploying database servers, 1074
  deploying on Mac OS X, 1063
libdbodbc11_r.so
  deploying database servers, 1074
libdbodbc11_r.so.1
  deploying on Unix, 1060
libdbodm11
  about, 469
  deploying ODBC clients, 1037
  Unix ODBC driver manager, 469
libdbodm11.dylib
  deploying on Mac OS X, 1063
libdbodm11.so
  deploying on Unix, 1060
libdbofsp_r.so
  deploying database utilities, 1082
libdbput11_r.dylib
  deploying on Mac OS X, 1063
libdbput11_r.so
  deploying on Unix, 1060
libdbrsa11.dylib
  RSA encryption, 1081
libdbrsa11.so
  RSA encryption, 1081
libdbrsa11_r.dylib
  RSA encryption, 1081
libdbrsa11_r.so
  deploying JDBC clients, 1046
  deploying Open Client, 1046
libdbrsa11_r.dll
  AES encryption, 1081
libdbsmtp11_r
  deploying SQL Remote, 1084
libdbtasks11
  deploying ODBC clients, 1037
libdbtasks11.dylib
  deploying database utilities, 1082
libdbtasks11.so
Index

deploying database utilities, 1082
deploying embedded SQL clients, 1045
libdbtasks11_r
  deploying ODBC clients, 1037
deploying SQL Remote, 1084
libdbtasks11_r.dylib
  deploying database servers, 1074
deploying database utilities, 1082
deploying on Mac OS X, 1063
libdbtasks11_r.so
  deploying database servers, 1074
deploying database utilities, 1082
libdbtasks11_r.so.1
  deploying on Unix, 1060
libdbtool11_r
  about, 948
deploying SQL Remote, 1084
libdbtool11_r.dylib
  deploying database utilities, 1082
deploying on Mac OS X, 1063
libdbtool11_r.so
  deploying database utilities, 1082
libdbtool11_r.so.1
  deploying on Unix, 1060
libsybbr600_r.dylib
  deploying on Mac OS X, 1063
libsybbr600_r.so.1
  deploying on Unix, 1060
libmljodbc.dylib
  deploying on Mac OS X, 1063
libmljodbc.so.1
  deploying on Unix, 1060
libraries
  calling external libraries from stored procedures or
  functions, 670
dblstm.lib, 950
dbtools11.lib, 950
embedded SQL, 534
using the import libraries, 950
library
  dblib11.lib, 534
dblibm.lib, 534
libdblib11.so, 534
libdblib11_r.so, 534
libdbtasks11.so, 534
libdbtasks11_r.so, 534
library functions
  embedded SQL, 593
  libsybbr.dll
  deploying database servers, 1074
  libsybbr.dylib
  deploying database servers, 1074
  libsybbr.so
  deploying database servers, 1074
licensing
  DBLicense function, 961
  web servers, 938
line length
  SQL preprocessor output, 590
line numbers
  SQL preprocessor, 590
Linked Servers
  InProcess option, 456
  OLE DB, 456
  RPC option, 456
  RPC Out option, 456
Linux
  deployment issues, 1020
directory structure, 1020
Listeners
  glossary definition, 1102
liveness
  connections, 605
LivenessTimeout property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 286
local temporary tables
  glossary definition, 1102
LocalOnly property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 436
lockedPreferences
  configurable option , 1071
locks
  glossary definition, 1103
log files
  glossary definition, 1103
log4j.jar
  deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
LogFile property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 286
logical indexes
  glossary definition, 1103
LONG BINARY data type
  embedded SQL, 582
retrieving in embedded SQL, 583
sending in embedded SQL, 584
LONG NVARCHAR data type
embedded SQL, 582
retrieving in embedded SQL, 583
sending in embedded SQL, 584
LONG VARCHAR data type
embedded SQL, 582
retrieving in embedded SQL, 583
sending in embedded SQL, 584
LONGBINARY data type
embedded SQL, 547
LONGNVARCHAR data type
embedded SQL, 547
LONGVARCHAR data type
embedded SQL, 547
lost updates
about, 52
LTM
glossary definition, 1103

M
Mac OS X
checking JRE version, 1050
deployment issues, 1020
directory structure, 1020
macros
_SQL_OS_WINDOWS, 536
main method
Java in the database, 87, 102
maintenance releases
glossary definition, 1103
management tools
dbtools, 947
manual commit mode
controlling, 60
implementation, 62
transactions, 60
materialized views
glossary definition, 1103
maximumDisplayedRows
configurable option, 1071
MaxPoolSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 286
MDAC
deploying, 1031
version, 1031
membership
result sets, 41
MergeModule.CABinet
deployment wizard, 1023
message log
glossary definition, 1104
Message property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 350, 357, 367
message stores
glossary definition, 1104
message systems
glossary definition, 1104
message types
glossary definition, 1104
messages
callback, 605
server, 605
MessageType property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 368
metadata
glossary definition, 1104
metadata support
installing for jConnect, 507
MetaDataCollections field [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 377
method signatures
Java, 707
Microsoft .NET
accessing web services tutorial, 868
using data types tutorial, 887
Microsoft Transaction Server
three-tier computing, 68
Microsoft Visual C++
embedded SQL support, 533
MIME type
web service, 932
MinPoolSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 287
mirror logs
glossary definition, 1104
mixed cursors
ODBC, 55
mldesign.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
mlmon.ini
deploying administration tools, 1048
mlplugin.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
mlstream.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
MobiLink
glossary definition, 1105
MobiLink clients
glossary definition, 1105
MobiLink Monitor
glossary definition, 1105
MobiLink server
glossary definition, 1105
MobiLink system tables
glossary definition, 1105
MobiLink users
glossary definition, 1105
mobilink.jar
deploying administration tools on Linux/Unix/Mac
OS X, 1065, 1069
deploying administration tools on Windows, 1054,
1057
MSDASQL
OLE DB provider, 448
msiexec
deployment wizard, 1023
silent install, 1027
msxml4.dll
deploying SQL Anywhere, 1077
multi-row fetches
ESQL, 577
multi-row inserts
ESQL, 577
multi-row puts
ESQL, 577
multi-row queries
cursors, 575
multi-threaded applications
embedded SQL, 557, 559
Java in the database, 102
ODBC, 464, 478
Unix, 468
multiple result sets
DESCRIBE statement, 588
ODBC, 496
myDispose method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 341
MyIP property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 436
N
name SQLDA field
about, 566
NativeError property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 350,
358, 368
natural joins
glossary definition, 1099
NCHAR data type
embedded SQL, 547
network protocols
glossary definition, 1105
network server
glossary definition, 1105
NewPassword property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 287
newsgroups
technical support, xvii
NEXT_CONNECTION function
deployment example, 940
NEXT_HTTP_HEADER function
web services, 919
NEXT_HTTP_VARIABLE function
web services, 917
NEXT_SOAP_HEADER function
web services, 925
NextResult method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 340
NFIXCHAR data type
embedded SQL, 547
NO SCROLL cursors
about, 39, 46
embedded SQL, 56
normalization
glossary definition, 1105
Notifiers
glossary definition, 1106
NotifyAfter property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 185
ntodbc.h
about, 466
compilation platform, 485
NULL
  dynamic SQL, 565
  indicator variables, 551
null-terminated string
  embedded SQL data type, 542
NVARCHAR data type
  embedded SQL, 547

O
  object trees
    glossary definition, 1106
object-oriented programming
  style, 88
objects
  storage format, 100
obtaining time values
  about, 132
ODBC
  64-bit considerations, 485
  autocommit mode, 60
  backward compatibility, 465
  compatibility, 465
  conformance, 464
  connecting with no data source, 1044
  controlling autocommit behavior, 60
cursor types, 39
cursors, 55, 491
data alignment, 489
data sources, 1042
driver deployment, 1037
error checking, 498
FETCH FOR UPDATE, 54
glossary definition, 1106
handles, 473
header files, 466
import libraries, 466
introduction, 464
introduction to programming, 6
linking, 466
multi-threaded applications, 478
multiple result sets, 496
prepared statements, 483
programming, 463
registry entries, 1042
result sets, 496
sample application, 474
sample program, 471
SQL statements, 24
SQLSetConnectAttr, 479
stored procedures, 496
Unix development, 468
version supported, 464
Windows Mobile, 467
ODBC Administrator
glossary definition, 1106
ODBC API
  about, 463
ODBC client
  deploying, 1037
  install, 1037
ODBC data sources
  deploying, 1041
glossary definition, 1106
ODBC driver
  components, 1037
  defining data sources, 1041
  deploying, 1037
  install, 1037
  registry settings, 1040
ODBC driver managers
  Unix, 469
ODBC drivers
  Unix, 468
ODBC programming interface
  introduction, 6
odbc.h
  about, 466
OEM.ini
  administration tools, 1071
Offset property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 389
OLE DB
  about, 448
  autocommit mode, 60
  controlling autocommit behavior, 60
cursor types, 39
cursors, 55
deploying, 1030
Microsoft Linked Server setup, 456
ODBC and, 448
provider deployment, 1030
supported interfaces, 457
supported platforms, 448
updates, 453
updating data through a cursor, 453
PersistSecurityInfo property [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 288
personal server
   deploying, 1082
glossary definition, 1107
PHP
   installing, 740
PHP functions
   sasql_affected_rows, 755
   sasql_close, 756
   sasql_commit, 756
   sasql_connect, 756
   sasql_data_seek, 757
   sasql_disconnect, 758
   sasql_error, 758
   sasql_errorcode, 759
   sasql_escape_string, 759
   sasql_fetch_array, 760
   sasql_fetch_assoc, 760
   sasql_fetch_field, 761
   sasql_fetch_object, 762
   sasql_fetch_row, 762
   sasql_field_count, 763
   sasql_field_seek, 763
   sasql_free_result, 764
   sasql_get_client_info, 764
   sasql_insert_id, 764
   sasql_message, 765
   sasql_multi_query, 765
   sasql_next_result, 766
   sasql_num_fields, 766
   sasql_num_rows, 767
   sasql_pconnect, 767
   sasql_prepare, 768
   sasql_query, 769
   sasql_real_escape_string, 770
   sasql_real_query, 770
   sasql_result_all, 771
   sasql_rollback, 772
   sasql_set_option, 772
   sasql_sqlstate, 783
   sasql_stmt_affected_rows, 773
   sasql_stmt_bind_param, 773
   sasql_stmt_bind_param_ex, 774
   sasql_stmt_bind_result, 775
   sasql_stmt_close, 775
   sasql_stmt_data_seek, 776
   sasql_stmt_errno, 776
   sasql_stmt_error, 777
   sasql_stmt_execute, 777
   sasql_stmt_fetch, 778
   sasql_stmt_field_count, 778
   sasql_stmt_free_result, 779
   sasql_stmt_insert_id, 779
   sasql_stmt_next_result, 780
   sasql_stmt_num_rows, 780
   sasql_stmt_param_count, 781
   sasql_stmt_reset, 781
   sasql_stmt_result_metadata, 781
   sasql_stmt_send_long_data, 782
   sasql_stmt_store_result, 782
   sasql_store_result, 783
   sasql_use_result, 784

PHP hypertext preprocessor
   about, 739

PHP keyword
   external environment, 715

PHP module
   about, 739
   API reference, 744
   configuring SQL Anywhere module, 744
   installing SQL Anywhere module, 741
   introduction to programming, 12
   running PHP scripts in web pages, 747
   versions, 741
   writing scripts, 748
   writing web pages, 746

php-5.1.[1-6]_sqlanywhere_extenv11.dll
   deploying, 1079

php-5.1.[1-6]_sqlanywhere_extenv11.so
   deploying, 1079

php-5.2.[0-6]_sqlanywhere_extenv11.dll
   deploying, 1079

php-5.2.[0-6]_sqlanywhere_extenv11.so
   deploying, 1079

phpenv.php
   deploying, 1079
   PHP support module, 716

PHPXC
   environment variable, 716
   physical indexes
      glossary definition, 1107
   place holders
      dynamic SQL, 561
   platforms
      cursors, 39
Java in the database support, 81
plug-in modules
glossary definition, 1107
plug-ins
deploying, 1049
policies
glossary definition, 1107
policy.11.0.iAnywhere.Data.SQLAnywhere.dll
deploying .NET clients, 1029
polling
glossary definition, 1107
pooling
collections with .NET Data Provider, 115
POOLING option
.NET Data Provider, 115
Pooling property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 288
positioned DELETE statement
about, 36
positioned UPDATE statement
about, 36
positioned updates
about, 34
PowerDesigner
glossary definition, 1107
PowerJ
glossary definition, 1107
Precision property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 288
predicates
glossary definition, 1107
prefetch
cursor performance, 50
cursors, 51
fetching multiple rows, 35
prefetch option
cursors, 51
PrefetchBuffer property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 289
PrefetchRows property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 289
Prepare method
using, 27
Prepare method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 228
PREPARE statement
about, 561
PREPARE TRANSACTION statement
and Open Client, 847
prepared statements
ADO.NET overview, 27
bind parameters, 27
cursors, 32
dropping, 27
JDBC, 519
ODBC, 483
Open Client, 844
using, 26
PreparedStatement interface
about, 519
prepareStatement method
JDBC, 28
preparing
statements, 26
to commit, 68
preprocessor
about, 532
running, 533
primary key constraints
glossary definition, 1108
primary keys
glossary definition, 1108
obtaining values for, 129
primary tables
glossary definition, 1108
println method
Java in the database, 86
ProcedureParameters field [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 377
procedures
embedded SQL, 586
ODBC, 496
parameters to web service clients, 908
refreshing from SQL Anywhere Explorer, 21
result sets, 586
Procedures field [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 378
program structure
embedded SQL, 536
ProgramData
deploying administration tools on Windows, 1056
programming interfaces
(see also APIs)
C API, 9
JDBC API, 7
ODBC API, 6
Perl DBD::SQLAnywhere API, 10
Python Database API, 11
Ruby APIs, 13
SQL Anywhere .NET API, 4
SQL Anywhere embedded SQL, 8
SQL Anywhere OLE DB and ADO APIs, 5
SQL Anywhere PHP DBI, 12
SQL Anywhere web services, 14
Sybase Open Client API, 15

properties
  - db_get_property function, 600

prototype
  - external functions, 673

providers
  - SQL Anywhere ASP.NET Health Monitoring Provider, 155
  - SQL Anywhere ASP.NET Membership Provider, 155
  - SQL Anywhere ASP.NET Profiles Provider, 155
  - SQL Anywhere ASP.NET Roles Provider, 155
  - SQL Anywhere ASP.NET Web Parts Personalization Provider, 155
  - supported in .NET, 110
  - supported in ASP.NET, 155

proxy tables
  - glossary definition, 1108

public fields
  - issues, 88

publication updates
  - glossary definition, 1108

publications
  - glossary definition, 1108

publisher
  - glossary definition, 1109

push notifications
  - glossary definition, 1109

push requests
  - glossary definition, 1109

PUT statement
  - modifying rows through a cursor, 36
  - multi-row, 577
  - wide, 577

Python
  - closing connections, 735
  - commit method, 736
  - creating connections, 735
  - creating cursors, 736
  - executing SQL statements, 736

inserting into tables, 736
installing sqlanydb on Unix and Mac OS X, 734
installing sqlanydb on Windows, 733
multiple inserts, 737
sqlanydb, 731
writing sqlanydb scripts, 735

Python Database API
  - introduction to programming, 11

Python Database support
  - about, 731

Q

qaagent.exe
  - deploying on Windows, 1051
qaconnector.jar
  - deploying on Mac OS X, 1063
  - deploying on Unix, 1060
  - deploying on Windows, 1051
QAnywhere
  - glossary definition, 1109
QAnywhere Agent
  - glossary definition, 1109
qanywhere.jpr
  - deploying administration tools on Linux/Unix/Mac OS X, 1066, 1069
  - deploying administration tools on Windows, 1054, 1057
qapliugin.jar
  - deploying on Mac OS X, 1063
  - deploying on Unix, 1060
  - deploying on Windows, 1051

queries
  - ADO Recordset object, 451, 452
  - glossary definition, 1109
  - single-row, 574

quick start
  - web services, 851

quotation marks
  - Java in the database strings, 86

quoted identifiers
  - sql_needs_quotes function, 613
QuotIdentifier method [SA .NET 2.0]
  - iAnywhere.Data.SQLAnywhere namespace, 239

R

Rails
  - about, 812
installing ActiveRecord adapter, 810
tutorial, 813

RDBMS
  glossary definition, 1110

Read method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 340
  read only
cursors, 39
deploying databases, 1078
read only cursors
about, 39

ReceiveBufferSize property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 437
record sets
  ADO programming, 453

RecordsAffected property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 316, 425
Recordset ADO object
  ADO, 451
  ADO programming, 454
  updating data, 453
Recordset object
  ADO, 452
recovery
distributed transactions, 71

Redirector
  glossary definition, 1109

reentrant code
  multi-threaded embedded SQL example, 557
reference databases
  glossary definition, 1109
referenced object
  glossary definition, 1110
referencing object
  glossary definition, 1110
referential integrity
  glossary definition, 1110
registering
  .NET Data Provider, 139
  DLLs for deployment, 1077
SQL Anywhere ASP.NET connection string, 158
SQL Anywhere ASP.NET providers, 159
registry
  deploying, 1075
  deploying administration tools on Windows, 1055
  ODBC, 1042
  Wow6432Node, 1055

registry settings
  ODBC driver, 1040
regular expressions
  glossary definition, 1110
remote databases
  glossary definition, 1110
REMOTE DBA authority
  glossary definition, 1110
remote IDs
  glossary definition, 1111
REMOTEPWD
  using, 509
Remove method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 204, 296, 413
RemoveAt method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 204
RemoveAt methods [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 413
RemoveAt(Int32) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 413
RemoveAt(String) method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 414
replication
glossary definition, 1111
Replication Agent
  glossary definition, 1111
replication frequency
glossary definition, 1111
replication messages
  glossary definition, 1111
Replication Server
glossary definition, 1111
reportErrors
  configurable option, 1071
request processing
  embedded SQL, 589
requests
  aborting, 597
requirements
  Open Client applications, 841
ReservedWords field [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 378
ResetCommandTimeout method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 229
ResetDbType method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 395
resource dispensers
three-tier computing, 68
resource managers
about, 66
three-tier computing, 68
Restrictions field [SA .NET 2.0]

diAnywhere.Data.SQLAnywhere namespace, 379
result sets
ADO Recordset object, 451, 452
cursors, 29
Java in the database stored procedures, 103
JDBC, 523
metadata, 58
multiple ODBC, 496
ODBC, 491, 496
Open Client, 846
retrieving ODBC, 493
stored procedures, 586
using, 32
web services, 905
Results method
JDBCExample, 523
retrieving
ODBC, 493
SQLDA, 571
RetryConnectionTimeout property [SA .NET 2.0]

diAnywhere.Data.SQLAnywhere namespace, 289
return codes
about, 1013
ODBC, 498
return types
external functions, 686
return values and result sets
web clients, 905
role names
glossary definition, 1111
roles
glossary definition, 1111
rollback logs
glossary definition, 1112
Rollback methods [SA .NET 2.0]

diAnywhere.Data.SQLAnywhere namespace, 443
ROLLBACK statement
cursors, 62
ROLLBACK TO SAVEPOINT statement
cursors, 63
Rollback() method [SA .NET 2.0]

diAnywhere.Data.SQLAnywhere namespace, 443
Rollback(String) method [SA .NET 2.0]
sqlany_new_connection function, 832
sqlany_num_cols function, 832
sqlany_num_params function, 833
sqlany_num_rows function, 833
sqlany_prepare function, 834
sqlany_rollback function, 835
sqlany_sqlstate function, 835
Ruby APIs
  introduction to programming, 13
Ruby DBI
  about, 814
  connection examples, 814
  installing dbd-sqlanywhere, 810
Ruby on Rails
  about, 812
  installing ActiveRecord adapter, 810
  tutorial, 813
RubyGems
  installing, 810
runtime classes
  Java in the database, 85

S
sa_config.csh file
  deployment, 1021
sa_config.sh file
  deployment, 1021
sa_external_library_unload
  using, 688
SA_GET_MESSAGE_CALLBACK_PARM
  SQLSetConnectAttr, 479
SA_REGISTER_MESSAGE_CALLBACK
  SQLSetConnectAttr, 479
SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK
  SQLSetConnectAttr, 479
SA_SQL_ATTR_TXN_ISOLATION
  SQLSetConnectAttr, 479
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
  isolation level, 491
SA_SQL_TXN_SNAPSHOT
  isolation level, 491
SA_SQL_TXN_STATEMENT_SNAPSHOT
  isolation level, 491
SABulkCopy class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 178
  SABulkCopy members [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 178
  SABulkCopy(SAConnection) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 179
  SABulkCopy(SAConnection, SABulkCopyOptions, SATransaction) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 181
  SABulkCopy(String) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 180
  SABulkCopy(String, SABulkCopyOptions) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 181
  SABulkCopyColumnMapping class [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 189
  SABulkCopyColumnMapping constructors [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 190
  SABulkCopyColumnMapping members [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 190
  SABulkCopyColumnMapping() constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 190
  SABulkCopyColumnMapping(Int32, Int32) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 191
  SABulkCopyColumnMapping(Int32, String) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 192
  SABulkCopyColumnMapping(String, Int32) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 192
  SABulkCopyColumnMapping(String, String) constructor [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 193
  SABulkCopyColumnMappingCollection class [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 196
  SABulkCopyColumnMappingCollection members [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 197
  SABulkCopyOptions enumeration [SA .NET 2.0]
    iAnywhere.Data.SQLAnywhere namespace, 205
sacapi.h
  C API, 621
sacapi_error_size constant
SQL Anywhere C API, 651
sacapidll.h
C API, 619
interface library, 618
SACommand class
about, 117
deleting data, 120
inserting data, 120
retrieving data, 117
updating data, 120
using, 27, 117
using in a Visual Studio project, 149, 153
SACommand class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 206
SACommand constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 208
SACommand members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 207
SACommand() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 208
SACommand(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 209
SACommand(String, SAConnection) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 209
SACommand(String, SAConnection) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 209
SACommand(String, SAConnection, SATransaction) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 209
SACommandStringBuilder class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 269
SACommandStringBuilder constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 273
SACommandStringBuilder members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 269
SACommandStringBuilder() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 273
SACommandStringBuilder(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 274
SAConnectionStringBuilder class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 269
SAConnectionStringBuilder constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 273
SAConnectionStringBuilder members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 269
SAConnectionStringBuilder() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 273
SAConnectionStringBuilder(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 274
SACommLinksOptionsBuilder class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 241
SACommLinksOptionsBuilder constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 240
SACommLinksOptionsBuilder() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 241
SACommLinksOptionsBuilder(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 240
SAConnection class
connecting to a database, 114
using in a Visual Studio project, 148, 152
SAConnection constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 246
SAConnection members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 248
SAConnection function
using in a Visual Studio project, 152
SAConnection members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 247
SAConnection() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 249
SAConnection(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 249
SAConnectionStringStringBuilder class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 269
SAConnectionStringStringBuilder constructors [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 273
SAConnectionStringStringBuilder(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 274
SAConnectionStringBuilderBase class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 291
SAConnectionStringBuilderBase members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 292
SADataAdapter
obtaining primary key values, 129
SADataAdapter class
about, 117
deleting data, 124
inserting data, 124
obtaining result set schema information, 128
retrieving data, 123
updating data, 124
Index

using, 123
SADataAdapter class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 299
SADataAdapter constructors [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 301
SADataAdapter members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 299
SADataAdapter() constructor [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 301
SADataAdapter(SACommand) constructor [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 302
SADataAdapter(String, SAConnection) constructor [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 303
SADataAdapter(String, String) constructor [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 303
SADataReader class
   using, 117
   using in a Visual Studio project, 149, 153
SADataReader class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 310
SADataReader members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 311
SADataSourceEnumerator class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 341
SADataSourceEnumerator members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 342
SADbType enumeration [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 343
SADbType property [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 391
SADefault class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 348
SADefault members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 348
SAError class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 349
SAError members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 350
SAErrorCollection class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 352
SAErrorCollection members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 353
SAException class
   using in a Visual Studio project, 149, 153
SAException class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 355
SAException members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 356
SAFactory class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 359
SAFactory members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 360
SAInfoMessageEventArgs class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 366
SAInfoMessageEventArgs members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 366
SAInfoMessageEventHandler delegate [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 369
saip11.jar
   deploying on Windows, 1051
SAIsolationLevel enumeration [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 370
SAIsolationLevel property [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 442
sajvm.jar
   deploying database servers, 1074
salib.jar
   deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
SAMessageType enumeration [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 371
SAMetaDataCollectionNames class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 372
SAMetaDataCollectionNames members [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 372
samples
   .NET Data Provider, 145
   building embedded SQL applications, 538
   DBTools program, 954
   embedded SQL, 539
   embedded SQL applications, 538
   ODBC, 471
   SimpleViewer, 167
   static cursors in embedded SQL, 540
samples-dir
   documentation usage, xiv
SAOLEDB
   OLE DB provider, 448
SAParameter class [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 382
SAParameter constructors [SA .NET 2.0]
   iAnywhere.Data.SQLAnywhere namespace, 384
SAParameter members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 382
SAParameter() constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 384
SAParameter(String, Object) constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 384
SAParameter(String, SADbType) constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 385
SAParameter(String, SADbType, Int32) constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 385
SAParameter(String, SADbType, Int32, ParameterDirection, Boolean, Byte, Byte, String, DataRowVersion, Object) constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 387
SAParameter(String, SADbType, Int32, String) constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 386
SAParameterCollection class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 396
SAParameterCollection members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 396
SAPermission class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 414
SAPermission constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 414
SAPermission members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 416
SAPermissionAttribute class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 417
SAPermissionAttribute constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 419
SAPermissionAttribute members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 417
SARowsCopied event [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 189
SARowsCopiedEventArgs class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 420
SARowsCopiedEventArgs constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 421
SARowsCopiedEventArgs members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 420
SARowsCopiedEventHandler delegate [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 422
SARowUpdatedEventArgs class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 423
SARowUpdatedEventArgs constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 424
SARowUpdatedEventArgs members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 425
SARowUpdatedEventHandler delegate [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 426
SARowUpdatingEventArgs class [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 427
SARowUpdatingEventArgs constructor [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 427
SARowUpdatingEventArgs members [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 428
sasql_affected_rows function (PHP)
  syntax, 755
sasql_close function (PHP)
  syntax, 756
sasql_commit function (PHP)
  syntax, 756
sasql_connect function (PHP)
  syntax, 756
sasql_disconnect function (PHP)
  syntax, 758
sasql_error function (PHP)
  syntax, 758
sasql_errorcode function (PHP)
  syntax, 759
sasql_escape_string function (PHP)
  syntax, 759
sasql_fetch_array function (PHP)
  syntax, 760
sasql_fetch_assoc function (PHP)
  syntax, 760
sasql_fetch_field function (PHP)
  syntax, 761
sasql_fetch_object function (PHP)
  syntax, 762
sasql_fetch_row function (PHP)
  syntax, 762
sasql_field_count function (PHP)
  syntax, 763
sasql_field_seek function (PHP)
syntax, 763
sasql_free_result function (PHP)
syntax, 764
sasql_get_client_info function (PHP)
syntax, 764
sasql_insert_id function (PHP)
syntax, 764
sasql_message function (PHP)
syntax, 765
sasql_multi_query function (PHP)
syntax, 765
sasql_next_result function (PHP)
syntax, 766
sasql_num_fields function (PHP)
syntax, 766
sasql_num_rows function (PHP)
syntax, 767
sasql_pconnect function (PHP)
syntax, 767
sasql_prepare function (PHP)
syntax, 768
sasql_query function (PHP)
syntax, 769
sasql_real_escape_string function (PHP)
syntax, 770
sasql_real_query function (PHP)
syntax, 770
sasql_result_all function (PHP)
syntax, 771
sasql_rollback function (PHP)
syntax, 772
sasql_set_option function (PHP)
syntax, 772
sasql_sqlstate function (PHP)
syntax, 783
sasql_stmt_affected_rows function (PHP)
syntax, 773
sasql_stmt_bind_param function (PHP)
syntax, 773
sasql_stmt_bind_param_ex function (PHP)
syntax, 774
sasql_stmt_bind_result function (PHP)
syntax, 775
sasql_stmt_close function (PHP)
syntax, 775
sasql_stmt_data_seek function (PHP)
syntax, 776
sasql_stmt_errno function (PHP)
syntax, 776
sasql_stmt_errno function (PHP)
syntax, 776
sasql_stmt_error function (PHP)
syntax, 777
sasql_stmt_execute function (PHP)
syntax, 777
sasql_stmt_fetch function (PHP)
syntax, 778
sasql_stmt_free_result function (PHP)
syntax, 779
sasql_stmt_insert_id function (PHP)
syntax, 779
sasql_stmt_next_result function (PHP)
syntax, 780
sasql_stmt_num_rows function (PHP)
syntax, 780
sasql_stmt_param_count function (PHP)
syntax, 781
sasql_stmt_reset function (PHP)
syntax, 781
sasql_stmt_result_metadata function (PHP)
syntax, 781
sasql_stmt_send_long_data function (PHP)
syntax, 782
sasql_stmt_store_result function (PHP)
syntax, 782
sasql_stmt_store_result function (PHP)
syntax, 783
sasql_use_result function (PHP)
syntax, 784
SATcpOptionsBuilder class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 429
SATcpOptionsBuilder builders [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 432
SATcpOptionsBuilder members [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 429
SATcpOptionsBuilder() constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 432
SATcpOptionsBuilder(String) constructor [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 433
SATransaction class
using, 136
SATransaction class [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 439
SATransaction members [SA .NET 2.0]
Save method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 444
savepoints
cursors, 63
sbgscedll
FIPS-approved RSA encryption, 1081
Scale property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 391
SCEdit600.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
schema
adding the SQL Anywhere ASP.NET provider, 157
Health Monitoring provider, 165
Membership provider, 161
Profile provider, 163
Roles provider, 162
Web Part Personalization provider, 164
schemas
glossary definition, 1112
scjview
deploying on Mac OS X, 1063
deploying on Unix, 1060
scjview.exe
deploying on Windows, 1051
scjview.ini
deploying administration tools, 1048
scRepository (see .scRepository600)
script versions
glossary definition, 1112
script-based uploads
glossary definition, 1112
scripts
glossary definition, 1112
SCROLL cursors
about, 39, 49
embedded SQL, 56
scrollable cursors
about, 35
JDBC support, 502
scvwen600.jar
deploying on Mac OS X, 1063
deploying on Unix, 1060
deploying on Windows, 1051
secured features
glossary definition, 1112
security
Java in the database, 105
security manager
about, 105
SELECT statement
dynamic, 563
single row, 574
SelectCommand property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 305
self-registering DLLs
deploying SQL Anywhere, 1077
SendBufferSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 437
sensitive cursors
about, 46
delete example, 42
embedded SQL, 56
introduction, 42
update example, 44
sensitivity
cursors, 41, 42
delete example, 42
isolation levels, 54
update example, 44
serialization
objects in tables, 100
server address
eMBEDDED SQL function, 600
Server Explorer
Visual Studio, 168
server management requests
glossary definition, 1112
server message stores
glossary definition, 1113
server side autocommit
about, 62
server-initiated synchronization
glossary definition, 1112
ServerName property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 290
ServerPort property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 437
servers
locating, 611
web services, 849
web services quick start, 851
ServerVersion property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 253
return codes, 1014
Source property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 351, 358, 368
SourceColumn property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 195, 393
SourceColumnNullMapping property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 393
SourceOrdinal property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 196
SourceVersion property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 394
sp_tsql_environment system procedure
  setting options for jConnect, 510
SQL
  ADO applications, 24
  applications, 24
  embedded SQL applications, 24
  glossary definition, 1113
  JDBC applications, 24
  ODBC applications, 24
  Open Client applications, 24
SQL Anywhere
  C API, 617
    documentation, xii
    glossary definition, 1113
SQL Anywhere .NET API
  about, 4
SQL Anywhere .NET Data Provider
  about, 109
    tutorial, 145
SQL Anywhere 11 Demo.dsn
  Windows Mobile ODBC, 467
SQL Anywhere ASP.NET data providers
  about, 155
SQL Anywhere C API
  a_sqlany_bind_param struct, 639
  a_sqlany_bind_param_info struct, 640
  a_sqlany_column_info struct, 641
  a_sqlany_data_direction enumeration, 647
  a_sqlany_data_info struct, 642
  a_sqlany_data_type enumeration, 648
  a_sqlany_data_value struct, 643
  a_sqlany_native_type enumeration, 650
  C API, 618
    connecting.cpp, 653
    dbcapri_isql.cpp, 654
examples, 653
  fetching_a_result_set.cpp, 657
  fetching_multiple_from_sp.cpp, 659
  interface library, 618
  preparing_statements.cpp, 661
  sacapi.h, 621
  sacapi_error_size constant, 651
  sacapidll.h file, 619
  send_retrieve_full_blob.cpp, 663, 665
  sqlany_affected_rows function, 621
  sqlany_bind_param function, 621
  sqlany_clear_error function, 622
  sqlany_client_version function, 622
  sqlany_commit function, 622
  sqlany_connect function, 623
  sqlany_current_api_version constant, 652
  sqlany_describe_bind_param function, 624
  sqlany_disconnect function, 624
  sqlany_error function, 625
  sqlany_execute function, 625
  sqlany_execute_direct function, 626
  sqlany_execute_immediate function, 627
  sqlany_fetch_absolute function, 627
  sqlany_fetch_next function, 628
  sqlany_finalize_interface function, 620
  sqlany_finil function, 628
  sqlany_free_connection function, 628
  sqlany_free_stmt function, 629
  sqlany_get_bind_param_info function, 629
  sqlany_get_column function, 629
  sqlany_get_column_info function, 630
  sqlany_get_data function, 631
  sqlany_get_data_info function, 631
  sqlany_get_next_result function, 632
  sqlany_init function, 633
  sqlany_initialize_interface function, 619
  sqlany_make_connection function, 633
  sqlany_new_connection function, 634
  sqlany_num_cols function, 634
  sqlany_num_params function, 635
  sqlany_num_rows function, 635
  sqlany_prepare function, 635
  sqlany_reset function, 636
  sqlany_rollback function, 637
  sqlany_send_param_data function, 637
  sqlany_sqlstate function, 638
  SQLAnywhereInterface, 644
SQL Anywhere Explorer
sqlany_make_connection function
C API, 633
sqlany_new_connection function
C API, 634
Ruby API, 832
sqlany_num_cols function
C API, 634
Ruby API, 832
sqlany_num_params function
C API, 635
Ruby API, 833
sqlany_num_rows function
C API, 635
Ruby API, 833
sqlany_prepare function
C API, 635
Ruby API, 834
sqlany_reset function
C API, 636
sqlany_rollback function
C API, 637
Ruby API, 835
sqlany_send_param_data function
C API, 637
sqlany_sqlstate function
C API, 638
Ruby API, 835
sqlanydb
about, 731
installing on Unix and Mac OS X, 734
installing on Windows, 733
Python Database API, 11
writing Python scripts, 735
sqlanywhere.jdbc
deploying administration tools on Linux/Unix/Mac OS X, 1065, 1068
deploying administration tools on Windows, 1054, 1056
sqlanywhere_commit function (deprecated)
syntax, 785
sqlanywhere_connect function (deprecated)
syntax, 785
sqlanywhere_data_seek function (deprecated)
syntax, 786
sqlanywhere_disconnect function (deprecated)
syntax, 787
sqlanywhere_en11.chm
deploying on Windows, 1051
sqlanywhere_error function (deprecated)
syntax, 788
sqlanywhere_errorcode function (deprecated)
syntax, 788
sqlanywhere_execute function (deprecated)
syntax, 789
sqlanywhere_fetch_array function (deprecated)
syntax, 790
sqlanywhere_fetch_field function (deprecated)
syntax, 791
sqlanywhere_fetch_object function (deprecated)
syntax, 792
sqlanywhere_fetch_row function (deprecated)
syntax, 793
sqlanywhere_free_result function (deprecated)
syntax, 794
sqlanywhere_identity function (deprecated)
syntax, 795
sqlanywhere_insert_id function (deprecated)
syntax, 795
sqlanywhere_num_fields function (deprecated)
syntax, 795
sqlanywhere_num_rows function (deprecated)
syntax, 796
sqlanywhere_pconnect function (deprecated)
syntax, 797
sqlanywhere_query function (deprecated)
syntax, 798
sqlanywhere_result_all function (deprecated)
syntax, 799
sqlanywhere_rollback function (deprecated)
syntax, 800
sqlanywhere_set_option function (deprecated)
syntax, 800
SQLAnywhereInterface
SQL Anywhere C API, 644
SQLBindCol ODBC function
about, 493
parameter size, 485
storage alignment, 489
SQLBindParam ODBC function
parameter size, 485
SQLBindParameter function
ODBC prepared statements, 27
prepared statements, 483
SQLBindParameter ODBC function
about, 482
parameter size, 485
storage alignment, 489
stored procedures, 496
SQLBrowseConnect ODBC function
about, 476
SQLBulkOperations
ODBC function, 36
SQLCA
about, 555
changing, 557
fields, 555
length of, 555
multiple, 559
threads, 557
sqlcabc SQLCA field
about, 555
sqlcaid SQLCA field
about, 555
sqlcode SQLCA field
about, 555
SQLColAttribute ODBC function
parameter size, 485
SQLColAttributes ODBC function
parameter size, 485
SQLConnect ODBC function
about, 476
SQLCOUNT
sqlerror SQLCA field element, 556
sqld SQLDA field
about, 566
SQLDA
about, 561, 565
allocating, 593
descriptors, 58
fields, 566
filling, 612
freeing, 612
host variables, 566
sqlllen field, 568
strings, 612
sqlda_storage function
about, 614
sqlda_string_length function
about, 614
sqldabc SQLDA field
about, 566
sqldaif SQLDA field
about, 566
sqldata SQLDA field
about, 566
SQLDATETIME data type
embedded SQL, 547
sqldef.h
data types, 542
sqldef.h file
software exit codes location, 1014
SQLDescribeCol ODBC function
parameter size, 485
SQLDescribeParam ODBC function
parameter size, 485
SQLDriverConnect ODBC function
about, 476
sqlerrd SQLCA field
about, 555
sqlerrmc SQLCA field
about, 555
sqlerrml SQLCA field
about, 555
SQLError ODBC function
about, 498
sqlerror SQLCA field
elements, 556
SQLCOUNT, 556
SQLIOCOUNT, 556
SQLIOESTIMATE, 557
sqlerror_message function
about, 614
sqlerrp SQLCA field
about, 555
SQLExecDirect ODBC function
about, 481
bound parameters, 482
SQLExecute function
ODBC prepared statements, 27
SQLExtendedFetch
ODBC function, 34, 35
SQLExtendedFetch ODBC function
about, 493
parameter size, 485
stored procedures, 496
SQLFetch
ODBC function, 34
SQLFetch ODBC function
about, 493
stored procedures, 496
Index

SQLFetchScroll
  ODBC function, 34, 35
SQLFetchScroll ODBC function
  about, 493
  parameter size, 485
SQLFreeHandle ODBC function
  using, 473
SQLFreeStmt function
  ODBC prepared statements, 27
SQLGetConnectAttr ODBC function
  about, 478
SQLGetData ODBC function
  about, 493
  parameter size, 485
  storage alignment, 489
SQLGetDescRec ODBC function
  parameter size, 485
sqlind SQLDA field
  about, 566
SQLI OCOUNT
  sqlerror SQLCA field element, 556
SQLIOESTIMATE
  sqlerror SQLCA field element, 557
SQLJ standard
  about, 78
sqlllen SQLDA field
  about, 566, 568
  DESCRIBE statement, 568
  describing values, 568
  retrieving values, 571
  sending values, 570
SQLLEN versus SQLINTEGER
  ODBC, 485
sqlname SQLDA field
  about, 566
SQLNumResultCols ODBC function
  stored procedures, 496
SQLParamOptions ODBC function
  parameter size, 485
sqlpp utility
  about, 532
  running, 533
  syntax, 590
SQLPrepare function
  about, 483
  ODBC prepared statements, 27
SQLPutData ODBC function
  parameter size, 485
SQLRETURN
  ODBC return code type, 498
SQLRowCount ODBC function
  parameter size, 485
SQLSetConnectAttr
  ODBC applications, 479
SQLSetConnectAttr ODBC function
  about, 477
  transaction isolation levels, 491
SQLSetConnectOption ODBC function
  parameter size, 485
SQLSetDescRec ODBC function
  parameter size, 485
SQLSetParam ODBC function
  parameter size, 485
SQLSetPos ODBC function
  about, 494
  parameter size, 485
SQLSetScrollOptions ODBC function
  parameter size, 485
SQLSetStmtAttr ODBC function
  cursor characteristics, 492
SQLSetStmtOption ODBC function
  parameter size, 485
SqlState property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 351
sqlstate SQLCA field
  about, 556
SQLTransact ODBC function
  about, 474
sqltype SQLDA field
  about, 566
  DESCRIBE statement, 568
SQLULEN versus SQLUINTEGER
  ODBC, 485
sqlvar SQLDA field
  about, 566
  contents, 566
sqlwarn SQLCA field
  about, 555
standard output
  Java in the database, 86
standards
  SQLJ, 78
starting
  databases using jConnect, 509
StartLine property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 290
State property
  .NET Data Provider, 115
State property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 254
StateChange event [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 268
statement handles
  ODBC, 473
statement-level triggers
  glossary definition, 1114
statements
  insert, 26
static cursors
  about, 46
  ODBC, 55
static SQL
  about, 561
stax-api-1.0.jar
  deploying on Mac OS X, 1063
  deploying on Unix, 1060
  deploying on Windows, 1051
steps for using cursors
  about, 32
stored functions
  calling external functions, 670
  parameters to web service clients, 908
stored procedures
  .NET Data Provider, 134
  calling external functions, 670
  creating in embedded SQL, 586
  executing in embedded SQL, 586
  glossary definition, 1114
  INOUT parameters and Java, 104
  Java in the database, 103
  OUT parameters and Java, 104
  parameters to web service clients, 908
  result sets, 586
string
  data type, 614
string literal
  glossary definition, 1114
strings
  blank padding of DT_NSTRING, 542
  blank padding of DT_STRING, 542
  Java in the database, 86
structure packing
  header files, 534
subqueries
  glossary definition, 1114
subscriptions
  glossary definition, 1114
support
  newsgroups, xvii
supported platforms
  OLE DB, 448
Sybase Central
  adding JAR files, 99
  adding Java classes, 98
  adding ZIP files, 99
  configuring for deployment, 1071
  deploying, 1048
  deploying on Linux and Unix, 1059
  deploying on Windows without InstallShield, 1049
  deployment considerations, 1049
  glossary definition, 1114
  installing jConnect metadata support, 507
  opening from Visual Studio, 19
Sybase EAServer (see EAServer)
Sybase Enterprise Application Studio
  executing SQL statements, 25
Sybase Open Client API
  about, 839
  sybasecentral600.jar
    deploying on Mac OS X, 1063
    deploying on Unix, 1060
    deploying on Windows, 1051
symlink
  deployment on Unix, 1020
synchronization
  glossary definition, 1115
SyncRoot property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 401
SYS
  glossary definition, 1115
system objects
  glossary definition, 1115
system requirements
  .NET Data Provider, 139
system tables
  glossary definition, 1115
system views
  glossary definition, 1115
System:Transactions
  using, 137
table adapter
  Visual Studio, 172
TableMappings property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 306
Tables field [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 379
TableViewer
  .NET Data Provider sample project, 111
TcpOptionsBuilder property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 244
TcpOptionsString property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 244
TDS property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 438
technical support
  newsgroups, xvii
temporary tables
  glossary definition, 1115
threaded applications
  Unix, 1020
threads
  Java in the database, 102
  multiple SQLCAs, 559
  multiple thread management in embedded SQL, 557
  ODBC, 464
  ODBC applications, 478
  Unix development, 468
three-tier computing
  about, 66
  architecture, 67
  Distributed Transaction Coordinator, 68
  distributed transactions, 67
  EAServer, 68
  Microsoft Transaction Server, 68
  resource dispensers, 68
  resource managers, 68
Time structure
  time values in .NET Data Provider, 132
Timeout property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 438
times
  obtaining with .NET Data Provider, 132
TimeSpan
  .NET Data Provider, 132
TIMESTAMP data type
  conversion, 842
topics
  graphic icons, xvi
ToString method [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 246, 352, 369, 395, 439
tracing
  .NET support, 141
transaction attribute
  component, 72
transaction coordinator
  EAServer, 72
transaction log
  glossary definition, 1115
transaction log mirror
  glossary definition, 1116
transaction processing
  using the .NET Data Provider, 136
Transaction property [SA .NET 2.0]
  iAnywhere.Data.SQLAnywhere namespace, 214
transactional integrity
  glossary definition, 1116
transactions
  ADO, 454
  application development, 60
  autocommit mode, 60
  choosing ODBC transaction isolation level, 491
  controlling autocommit behavior, 60
  cursors, 62
  distributed, 66, 70
  glossary definition, 1115
  isolation level, 62
  ODBC, 474
  OLE DB, 454
TransactionScope class
  using, 137
transmission rules
  glossary definition, 1116
triggers
  glossary definition, 1116
troubleshooting
  cursor positioning, 34
  Java in the database methods, 102
  newsgroups, xvii
truncation
  FETCH statement, 553
  indicator variables, 553
  on FETCH, 553
TryGetValue method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 298
tutorials
accessing web services from JAX-WS, 871
developing a .NET database application, 167
SQL Anywhere .NET Data Provider, 145
using data types with JAX-WS, 892
using the .NET Data Provider Simple code sample, 147
using the .NET Data Provider Table Viewer code sample, 150
two-phase commit
and Open Client, 847
three-tier computing, 67, 68

U
ulplugin.jar
deploying on Windows, 1051
UltraLite
glossary definition, 1116
UltraLite runtime
glossary definition, 1116
ultralite.jpr
deploying administration tools on Windows, 1054, 1057
unchained mode
controlling, 60
implementation, 62
transactions, 60
Unconditional property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 290
Unicode
linking ODBC applications for Windows Mobile, 468
unique constraints
glossary definition, 1116
unique cursors
about, 39
Unix
deployment issues, 1020
directory structure, 1020
multi-threaded applications, 1020
ODBC, 468
ODBC driver managers, 469
unixodbc.h
about, 466
compilation platform, 485
unload
glossary definition, 1117
unload database utility [dbunload] (see unload utility [dbunload])
unload utility [dbunload]
deployment considerations, 1083
deployment for pre 10.0.0 databases, 1083
UnquoteIdentifier method [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 239
UNSIGNED BIGINT data type
embedded SQL, 547
UPDATE statement
positioned, 36
UpdateBatch ADO method
ADO programming, 454
updating data, 454
UpdateBatchSize property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 306
UpdateCommand property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 307
UpdatedRowSource property [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 215
updates
cursor, 453
upgrade database wizard
installing jConnect metadata support, 507
upgrade utility [dbupgrad]
installing jConnect metadata support, 507
uploads
glossary definition, 1117
URL path
about, 862
URL searchpart
about, 863
URL session ID
HTTP session, 935
URLs
database, 509
default services, 878
handling, 877
iAnywhere JDBC driver, 505
interpreting, 861
jConnect, 509
user-defined data types
glossary definition, 1117
UserDefinedTypes field [SA .NET 2.0]
iAnywhere.Data.SQLAnywhere namespace, 380
UserID property [SA .NET 2.0]
MIME types, 932
NEXT_HTTP_HEADER function, 919
NEXT_HTTP_VARIABLE function, 917
NEXT_SOAP_HEADER function, 925
quick start, 851
request handlers, 877
SOAP headers, 925
SOAP_HEADER function, 925
variables, 917
wide fetches
about, 35
ESQL, 577
wide inserts
ESQL, 577
JDBC, 521
wide puts
ESQL, 577
window (OLAP)
glossary definition, 1117
Windows
deploying administration tools without InstallShield, 1049
glossary definition, 1117
OLE DB support, 448
Windows Mobile
dbtool11.dll, 948
glossary definition, 1117
Java in the database unsupported, 81
ODBC, 467
OLE DB support, 448
WITH HOLD clause
cursors, 34
work tables
cursor performance, 50
glossary definition, 1117
working with procedures and functions using the SQL Anywhere Explorer
about, 21
WriteToServer methods [SA .NET 2.0]
WriteToServer(DataRow[]) method [SA .NET 2.0]
WriteToServer(DataTable) method [SA .NET 2.0]
WriteToServer(DataTable, DataRowState) method [SA .NET 2.0]
WriteToServer(IDataReader) method [SA .NET 2.0]

X
XML server
about, 849
creating web services, 856
XML services
listening for SOAP HTTP requests, 859
XML web server
quick start, 851