


CHAPTER 2

Connecting to a Database

About this chapter

This chapter describes how client applications connect to databases. It contains information about connecting to databases from ODBC applications and application development systems, as well as from Embedded SQL applications.

 For information on connecting to a database from Sybase Open Client applications, see "Adaptive Server Anywhere as an Open Server" on page 815. For information on connecting via JDBC, see "Data Access Using JDBC" on page 503.

Contents

Topic	Page
Introduction to connections	32
Simple connection examples	35
Working with ODBC data sources	42
Connection parameters	46
Troubleshooting connections	49
Using integrated logins	58

Introduction to connections

How connections are established

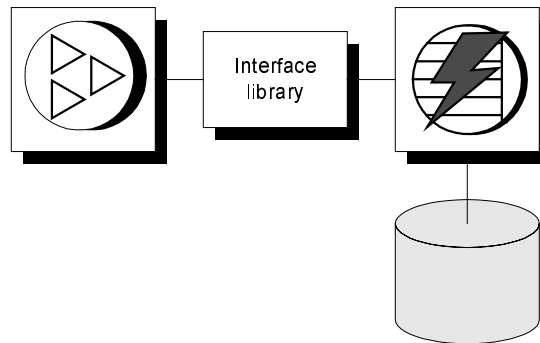
Any client application that uses a database must establish a **connection** to that database before any work can be done.

Once the connection is established, it forms a channel through which all of your activity from the client application takes place. For example, your permissions to carry out actions on the database are determined by your user ID—and the database server has your user ID because it is part of the request to establish a connection.

To establish a connection, the client application calls functions in one of the Adaptive Server Anywhere interfaces. Adaptive Server Anywhere provides the following interfaces:

- ◆ **ODBC** ODBC connections are discussed in this chapter.
- ◆ **Embedded SQL** Embedded SQL connections are discussed in this chapter.
- ◆ **Sybase Open Client** Open Client connections are not discussed in this chapter. For information on connecting from Open Client applications, see "Adaptive Server Anywhere as an Open Server" on page 815.
- ◆ **JDBC** JDBC connections are not discussed in this chapter. For information on connecting via JDBC, see "Data Access Using JDBC" on page 503.

The interface uses connection information included in the call from the client application, perhaps together with information held on disk in a file data source, to locate and connect to a server running the required database. The following figure is a simplified representation of the pieces involved.



What to read	If you want...	Consider reading...
	Some examples to get started quickly	"Simple connection examples" on page 35.
	To learn about data sources	"Working with ODBC data sources" on page 42
	To learn what connection parameters are available	"Connection parameters" on page 46.
	To see an in-depth description of how connections are established	"Troubleshooting connections" on page 49.

Connection parameters specify connections

When an application connects to a database, it uses a set of **connection parameters** to define the connection. Connection parameters include information such as the server name, the database name, and a user ID.

Each connection parameter is specified as a keyword-value pair, of the form *parameter=value*. For example, the password connection parameter for the default password is specified as follows:

```
Password=sql
```

Connection parameters are assembled into **connection strings**. In a connection string, each connection parameter is separated by a semicolon, as follows:

```
ServerName=asademo;DatabaseName=asademo
```

Representing connection strings

This chapter has many examples of connection strings. For ease of reading, connection parameters in this chapter are generally represented in the following form:

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

This is equivalent to the following connection string:

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

A connection string must be entered on a single line, with the parameter settings separated by semicolons.

Connection parameters are passed as connection strings

Connection parameters are passed to the interface library as a **connection string**. This string consists of a set of parameters, separated by semicolons:

```
parameter1=value;parameter2=value2;...
```

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

Many of the Adaptive Server Anywhere utilities accept a connection string as the `-c` command-line option and pass the connection string on to the interface library without change. For example, the following is a typical Collation utility (*dbcollat*) command line (which should be entered all on one line):

```
dbcollat -c "uid=dba;pwd=sql;dbn=asademo"  
c:\temp\asademo.col
```

Interactive SQL connection strings


Interactive SQL processes the connection string internally. These utilities do not simply pass on the connection parameters to the interface library. Do not use Interactive SQL to test command strings from a command prompt.

Simple connection examples

Although the connection model for Adaptive Server Anywhere is configurable, and can become complex, in many cases connecting to a database is very simple.

Who should read this section?

This section describes some simple cases of applications connecting to an Adaptive Server Anywhere database. When you are getting started, this section may be all you need.

 For more detailed information on available connection parameters and their use, see "Connection parameters" on page 46.

Connecting to the sample database from Interactive SQL

Many examples and exercises throughout the documentation start by connecting to the sample database from Interactive SQL. Here is how to carry out this step:

❖ To connect to the sample database from Interactive SQL:

- 1 **Start Interactive SQL** You can do this as follows:
 - ◆ From the Windows 95 or Windows NT Start menu, choose Sybase ► Adaptive Server Anywhere ► Interactive SQL.
 - ◆ In Windows 3.x, chose Interactive SQL from the Adaptive Server Anywhere program group.
 - ◆ Type **dbisql** at a system command prompt.

A connection window is displayed.
- 2 **Connect** From the ODBC data source list, select ASA 6.0 Sample. You can leave all the other fields empty. Click OK to start the database server and connect to the database.

Connecting to an embedded database

An **embedded database** is designed for use by a single application, runs on the same machine as the application, and is largely hidden from the application user.

When an application uses an embedded database, the personal server is generally not running when the application connects. In this case, you can start the database using the connection string. You can do this in one of the following ways:

- ◆ Specify the database file in the DatabaseFile (DBF) parameter of the connection string.

Using the DBF parameter

The following connection parameters show how the sample database could be loaded as an embedded database:

```
dbf=path\asademo.db
uid=dba
pwd=sql
```

where *path* is the name of your Adaptive Server Anywhere installation directory.

The DBF parameter specifies the database file to be used. If no server is running, one is started. If one or more servers are running, the database is loaded onto the default server.

When there are no more connections to the database (generally when the application that started it disconnects) the database is unloaded. If the server was started by the connection, it is stopped once the database is unloaded.

Using the Start parameter

The following connection parameters show how you can customize the startup of the sample database as an embedded database. This is useful if you wish to use command-line options, such as the cache size:

```
Start=dbeng6 -c 8M
dbf=path\asademo.db
uid=dba
pwd=sql
```

Extra cache needed for Java

If you are using Java in an embedded database, you should use the start line to provide more than the default cache size. For development purposes, a cache size of 8 Mb is sufficient.

Example: connecting from Interactive SQL

In this example, the sample database is used as an embedded database within Interactive SQL. This example assumes that you have no SQLCONNECT environment variable set.

❖ To connect to an embedded database from Interactive SQL:

- 1 Start Interactive SQL with no databases running. You can use either of the following ways:
 - ◆ From the Windows 95 or Windows NT Start menu, choose Sybase ► Adaptive Server Anywhere ► Interactive SQL.
 - ◆ In Windows 3.x, chose Interactive SQL from the Adaptive Server Anywhere program group.
 - ◆ Type **dbisql** at a system command prompt.

When Interactive SQL starts, it is not connected to any database.

- 2 Type **CONNECT** in the command window, and press F9 to execute the command. The connection dialog displays.
- 3 If you have an ODBC data source for your database, you can select that data source
- 4 Enter **DBA** as the user ID and **SQL** as the password. Then click the Database tab. Enter the full path of the sample database in the Database File field. For example, if your installation directory is *c:\sybase\asa6* you should enter the following:

```
c:\sybase\asa6\asademo.db
```
- 5 Leave all other fields blank, and click OK. Adaptive Server Anywhere starts up and loads the sample database, and Interactive SQL connects to the database.

Connecting using a data source

You can save sets of connection parameters in a **data source**. Data sources can be used by ODBC and Embedded SQL applications. You can create data sources from the ODBC Administrator.

Here, we show how to connect to the sample database from Interactive SQL using a data source

❖ To connect using a data source:

- 1 Start Interactive SQL with no databases running. You can use either of the following ways:
 - ◆ Type **dbisql** at a system command prompt, or
 - ◆ From the Windows 95 or Windows NT Start menu, choose Sybase►Adaptive Server Anywhere►Interactive SQL.
 - ◆ In Windows 3.x, chose Interactive SQL from the Adaptive Server Anywhere program group.

When Interactive SQL starts, it is not connected to any database.

- 2 Type **CONNECT** in the command window, and press F9 to execute the command. The connection dialog displays.
- 3 Enter **DBA** as the user ID and **SQL** as the password. Select **ASA 6.0 sample** from the drop-down list of ODBC data sources.

- 4 Leave all other fields blank, and click OK. Adaptive Server Anywhere starts up and loads the sample database, and Interactive SQL connects to the database.


The sample data source

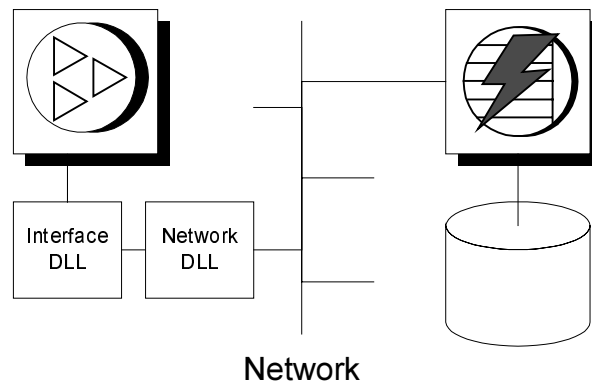
The ASA 6.0 Sample data source holds a set of connection parameters, including the database file and a Start parameter to start the database.

Connecting to a server on a network

To connect to a database running on a network server somewhere on a local or wide area network, the client software must be able to locate the database server. Adaptive Server Anywhere provides a network library (a DLL or shared library) that handles this task.

Any network connection is made over a network protocol. Several protocols are supported, including TCP/IP, IPX, and NetBIOS.

 For a full description of client/server communications over a network, see "Client/Server Communications" on page 685.



Specifying the server

Adaptive Server Anywhere server names must be unique on a local domain for a given network protocol. The following connection parameters provide a simple example for connecting to a server running elsewhere on a network:

```
eng=svr_name
dbn=db_name
uid=user_id
pwd=password
CommLinks=all
```

The client library first looks for a personal server of the given name, and then looks on the network for a server of the specified name.

☞ The above example finds any server started using default port number. However, you can start servers using other port numbers and in this case you need to provide more information in the CommLinks parameter. For information, see "CommLinks connection parameter" on page 43 of the book *Adaptive Server Anywhere Reference Manual*.

Specifying the protocol

If several protocols are available, you can instruct the network library which ones to use to improve performance. The following parameters use only the TCP/IP protocol:

```
eng=svr_name
dbn=db_name
uid=user_id
pwd=password
CommLinks=tcPIP
```

The network library searches for a server by broadcasting over the network. This can be a time-consuming process. Once a server is located, its name and network address are stored by the client library in a file. This entry is reused for subsequent connection attempts to that server using the specified protocol. This can make subsequent connections many times faster than a connection achieved by broadcast.

☞ Many other connection parameters are available to assist Adaptive Server Anywhere in locating a server efficiently over a network. For more information see "Network communications parameters" on page 54 of the book *Adaptive Server Anywhere Reference Manual*.

Using default connection parameters

Many connection parameters can be left unspecified, using default behavior to make a connection. You should be cautious about relying on default behavior in production environments, especially if you distribute your application to customers who may install other Adaptive Server Anywhere applications on their machine.

Default database server and database

If a single personal server is running, and it has loaded a single database, you can connect using entirely default parameters:

```
uid=user_id
pwd=password
```

Default database server

If more than one database is loaded on a single personal server, you need to specify the database you wish to connect to, but you can leave the server as a default:

```
dbn=db_name
uid=user_id
pwd=password
```

Default database

If more than one server is running, you need to specify which one you wish to connect to. If only one database is loaded on that server, you do not need to specify the database name. The following connection string connects to a named server, using the default database:

```
eng=server_name  
uid=user_id  
pwd=password
```

No defaults

The following connection string connects to a named server, using a named database:

```
eng=server_name  
dbn=db_name  
uid=user_id  
pwd=password
```

☞ For more information about default behavior, see "Troubleshooting connections" on page 49.

Connecting from Adaptive Server Anywhere utilities

All Adaptive Server Anywhere database utilities that communicate with the server (rather than acting directly on database files) do so using Embedded SQL. They follow the procedure outlined in "Troubleshooting connections" on page 49 when connecting to a database.

How database
tools obtain
connection
parameter values

Many of the administration utilities obtain the values of the connection parameters in the following way:

- 1 If there are values specified on the command line, those values are used for the connection parameters. For example, the following command starts a backup of the default database on the default server using the user ID **DBA** and the password **SQL**:

```
dbbackup -c "uid=dba;pwd=sql" c:\backup
```

- 2 If any command line values are missing, the application looks at the setting of the SQLCONNECT environment variable. This variable is not set automatically by Adaptive Server Anywhere.

☞ For a description of the SQLCONNECT environment variable, see "Environment variables" on page 5 of the book *Adaptive Server Anywhere Reference Manual*.

- 3 If parameters are not set in the command line, or the SQLCONNECT environment variable, then the application prompts for a user ID and password to connect to the default database on the default server.

☞ For a description of command line switches for each database tool, see chapter "Database Administration Utilities" on page 63 of the book *Adaptive Server Anywhere Reference Manual*.

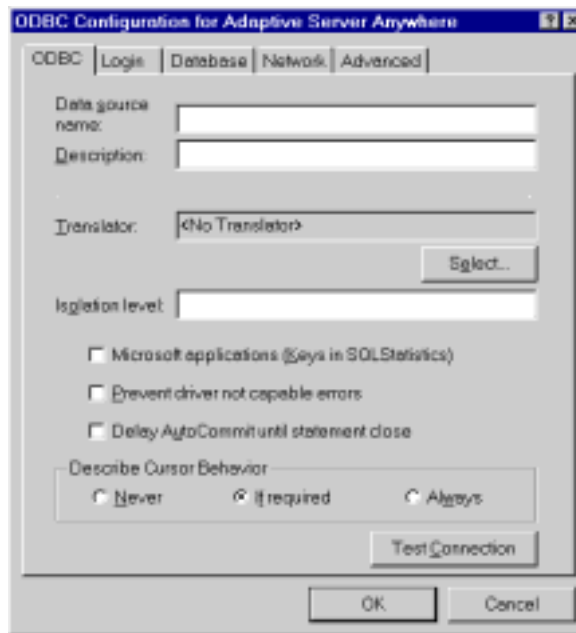
Working with ODBC data sources

	<p>The Open Database Connectivity (ODBC) interface is defined by Microsoft Corporation, and is a standard interface for connecting client applications to database management systems in the Windows and Windows NT environments. Many client applications, including application development systems, use the ODBC interface to access a wide range of database systems.</p> <p>Adaptive Server Anywhere can use ODBC data sources on UNIX as well as on Windows operating systems. For information, see "Using ODBC data sources on UNIX" on page 44.</p>
Where data sources are held	<p>When you connect to a database using ODBC, you do so using an ODBC data source. The data source contains a set of connection parameters. You need an ODBC data source on the client computer for each database to which you wish to connect.</p> <p>You can store a set of Adaptive Server Anywhere connection parameters as an ODBC data source. Data sources can be stored in the system registry or as files.</p> <p>If you have a data source, your connection string can simply name the data source to use:</p> <ul style="list-style-type: none">◆ Data source You can reference a data source in the registry using the DSN connection parameter: <code>DSN=my data source</code>◆ File data source You can reference a data source held in a file using the FileDSN connection parameter: <code>FileDSN=mysource.dsn</code>
Embedded SQL can use data sources	<p>Embedded SQL applications such as Interactive SQL and the other Adaptive Server Anywhere database administration utilities can also use ODBC data sources, even though they are not ODBC applications.</p>

Creating an ODBC data source

- For Windows operating systems, the ODBC Administrator provides a central place for managing ODBC data sources. You can start the ODBC Administrator from your Adaptive Server Anywhere program group.
- ❖ **To create an ODBC data source:**
 - 1 Start the ODBC Administrator. From the User DSN tab, click Add.

- 2 Select Adaptive Server Anywhere 6.0 from the list of drivers, and click FINISH. The Adaptive Server Anywhere ODBC Configuration window is displayed.



Many of the fields in this window are optional. Click the question mark at the top right of the window and click an entry field to find more information about that field.

- 3 When you have specified the parameters you need, click OK to close the window and create the data source.

You can edit a data source by selecting a data source from the list in the ODBC administrator window and clicking Configure.

Creating a File Data Source

Data sources are stored in the system registry. File data sources are an alternative, which are stored as files. File data sources typically have the extension *.dsn*. They consist of sections, each section starting with a name enclosed in square brackets. DSN files are very similar in layout to initialization files.

File data sources
can be distributed

One benefit of file data sources is that you can distribute the file to users. If the file is placed in the default location for file data sources, it is picked up automatically by ODBC. In this way, managing connections for many users can be made simpler.

Embedded SQL applications can also use ODBC file data sources.

❖ **To create an ODBC file data source:**

- 1 Start the ODBC Administrator. From the File DSN tab, click Add.
- 2 Select Adaptive Server Anywhere 6.0 from the list of drivers, and click Next.
- 3 Follow the instructions to create the data source

Using ODBC data sources on UNIX

On UNIX operating systems, ODBC data sources are held in a file named *.odbc.ini*. A sample file is as follows:

```
[My Data Source]
ENG=myserver
CommLinks=tcPIP
UID=dba
PWD=sql
```

The database server looks for this file as follows:

- ◆ The ODBCINI environment variable
- ◆ The ODBCHOME and HOME environment variables
- ◆ The user's home directory
- ◆ The path.

Using ODBC data sources on Windows CE

Windows CE does not provide an ODBC driver manager or an ODBC Administrator. On this platform, Adaptive Server Anywhere uses ODBC data sources stored in files. You can specify either the DSN or the FileDSN keyword to use these data source definitions—on Windows CE (only) these are synonyms.

Data source
location


The data source files are searched for in the following locations:

- 1 The directory from which the ODBC driver (*dbodbc6.dll*) was loaded. This is usually the Windows directory.

- 2 The directory specified in Location key of the Adaptive Server Anywhere section of the registry. This is usually the same as the Adaptive Server Anywhere installation directory. The default installation directory is:

```
\Program Files  
  \Adaptive Server Anywhere 6.0  
    \Windows
```

Each data source itself is held in a file. The file has the same name as the data source, with an extension of *.dsn*.

 For more information about file data sources, see "Creating a File Data Source" on page 43.

Connection parameters

The Adaptive Server Anywhere connection parameters are listed in the following table.

☞ For a full description of each of these connection parameters, see "Connection and Communication Parameters" on page 39 of the book *Adaptive Server Anywhere Reference Manual*.

Parameter	Short form	Argument
Agent	Agent	String (Any or Server)
AutoStart	AStart	Boolean
Autostop	AStop	Boolean
CommBufferSize	CBSize	Integer
CommBufferSpace	CBSpace	Integer
CommLinks	Links	String
ConnectionName *	CON	String
DatabaseFile	DBF	String
DatabaseName	DBN	String
DatabaseSwitches	DBS	String
DataSourceName	DSN	String
Debug	DBG	Boolean
DisableMultiRowFetch	DMRF	Boolean
EncryptedPassword	ENP	Encrypted string
Encryption	ENC	Boolean
EngineName / ServerName	ENG	String
FileDataSourceName	FileDSN	String
Integrated	INT	Boolean
LivenessTimeout	LTO	Integer
Logfile	LOG	String
Password **	PWD	String
StartLine	Start	String
Unconditional	UNC	Boolean
Userid **	UID	String

* Not used by ODBC connections

** Verbose form of keyword not used by ODBC connection parameters

Notes

- ◆ **Boolean values** Boolean (true or false) arguments are either YES, ON, 1, or TRUE if true, or NO, OFF, 0, or FALSE if false.
- ◆ **Case sensitivity** Connection parameters and their values are case insensitive.
- ◆ The connection parameters used by the interface library can be obtained from the following places (in order of precedence):
 - ◆ **Connection string** Parameters can be passed explicitly in the connection string.
 - ◆ **SQLCONNECT environment variable** Connection parameters can be stored in the SQLCONNECT environment variable.
 - ◆ **Data sources** Parameters can be stored in ODBC data sources.
 - ◆ **Compatibility data source** A special data source is available for compatibility with SQL Anywhere Version 5 client/server connections.
- ◆ **Character set restrictions** The server name must be composed of characters in the range 1 to 127 of the ASCII character set. There is no such limitation on other parameters.

For more information on the character set issues, see "Connection strings and character sets" on page 299.
- ◆ **Priority** The following rules govern the priority of parameters:
 - ◆ The entries in a connect string are read left to right. If the same parameter is specified more than once, the last one in the string is used.
 - ◆ If a string contains a data source or file data source entry, the profile is read from the configuration file, and the entries from the file are used if they are not already set. That is, if a connection string contains a data source name and sets some of the parameters contained in the data source explicitly, then in case of conflict the explicit parameters are used.

Connection parameter priorities

Connection parameters often provide more than one way of accomplishing a given task. This is particularly the case with embedded databases, where a database server is started by the connection string.

For example, if your connection will start a database, you can specify the database name using the DBN connection parameter or using the DBS parameter.

Here are some recommendations and notes for situations where connection parameters conflict:

- ◆ **Specify database files using DBF** You can specify a database file on the Start parameter or using the DBF parameter. It is recommended that you use the DBF parameter.
- ◆ **Specify database names using DBN** You can specify a database name on the Start parameter, the DBS parameter, or using the DBN parameter. It is recommended that you use the DBN parameter.
- ◆ **Use the Start parameter to specify cache size** Even though you use the DBF connection parameter to specify a database file, you may still want to tune the way in which it starts. You can use the Start parameter to do this.

A particular example is if you are using the Java features of Adaptive Server Anywhere. In that case, you should provide additional cache memory on the Start parameter. The following sample set of embedded database connection parameters describes a connection that may use Java features:

```
DBF=path\asademo.db
DBN=Sample
ENG=Sample Server
UID=dba
PWD=sql
Start=dbeng6 -c 8M
```

Troubleshooting connections

Who needs to read this section?

In many cases, establishing a connection to a database is straightforward using the information presented in the first part of this chapter.

However, if you are having problems establishing connections to a server, you may need to understand the process by which Adaptive Server Anywhere establishes connections in order to resolve your problems. This section describes how Adaptive Server Anywhere connections work.

Only read it if you need it

If you have no problem establishing connections to your database, you do not need to read this section.

The procedure the software follows is exactly the same for each of the following types of client application:

- ◆ **ODBC** Any ODBC application using the **SQLDriverConnect** function, which is the common method of connection for ODBC applications. Many application development systems, such as Powersoft PowerBuilder and Power++, belong to this class of application.
- ◆ **Embedded SQL** Any client application using Embedded SQL and using the recommended function for connecting to a database (**db_string_connect**).

The SQL CONNECT statement is available for Embedded SQL applications and in Interactive SQL. It has two forms: CONNECT AS... and CONNECT USING... The CONNECT USING statement uses **db_string_connect**. All the database administration tools, including Interactive SQL, use **db_string_connect**.

The steps in establishing a connection

In order to establish a connection, Adaptive Server Anywhere carries out the following steps:

- 1 **Locate the interface library** The client application must locate the ODBC driver or Embedded SQL interface library.
- 2 **Assemble a list of connection parameters** Connection parameters may be provided in several places, such as data sources, a connection string assembled by the application, and an environment variable. The parameters are assembled into a single list.

- 3 **Locate a server** Using the connection parameters, a database server must be located on your machine or over a network.
- 4 **Locate the database** Once the server is located, the database to which you wish to connect must be located.
- 5 **Start a personal server** If no server can be located, Adaptive Server Anywhere will attempt to start a personal database server and load the database.

Each of these steps is described in detail in the following sections.

Locating the interface library

The client application makes a call to one of the Adaptive Server Anywhere interface libraries. In general, the location of this DLL or shared library is transparent to the user. Here we describe how the library is located, in case of problems.

ODBC driver
location

For ODBC, the interface library is also called an ODBC **driver**. An ODBC client application calls the ODBC driver manager, and the driver manager locates the Adaptive Server Anywhere driver.

The ODBC driver manager looks in the supplied data source in the *odbc.ini* file or registry to locate the driver. When you create a data source using the ODBC Administrator, Adaptive Server Anywhere fills in the current location for your ODBC driver.

Embedded SQL
interface library
location

Embedded SQL applications call the interface library by name. The name of the Adaptive Server Anywhere Embedded SQL interface library is as follows:

- ◆ **Windows NT and Windows 95** *dblib6.dll*
- ◆ **Windows 3.x** *dblib6w.dll*
- ◆ **UNIX** *dblib6* with an operating-system-specific extension.

The locations that are searched depend on the operating system:

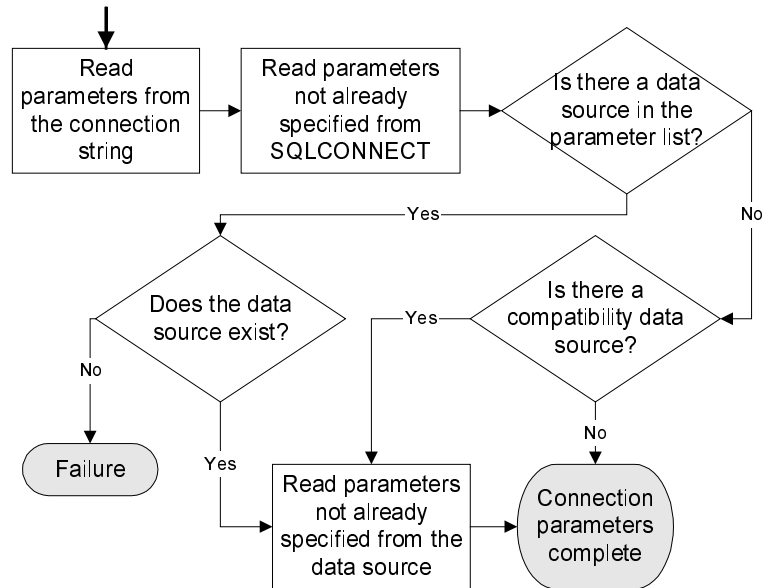
- ◆ **PC operating systems** In PC operating systems such as Windows and Windows NT, files are looked for in the current directory, in the system path, and in the *Windows* and *Windows\system* directories.
- ◆ **UNIX operating systems** In UNIX, files are looked for in the system path and the user path.

When the library is located

Once the interface library is located, a connection string is passed to it. The interface library uses the connection string to assemble a list of connection parameters, which it uses to establish a connection to a server. The following section describes how the list of connection parameters is assembled.

Assembling a list of connection parameters

The following figure illustrates how the interface libraries assemble the list of connection parameters they will use to establish a connection.



Notes

Key points from the figure are as follows:

- ◆ **Precedence** Parameters held in more than one place are subject to the following order of precedence:

Connection string > SQLCONNECT > data source

That is, if a parameter is supplied in a data source and in a connection string, the connection string value overrides the data source value.

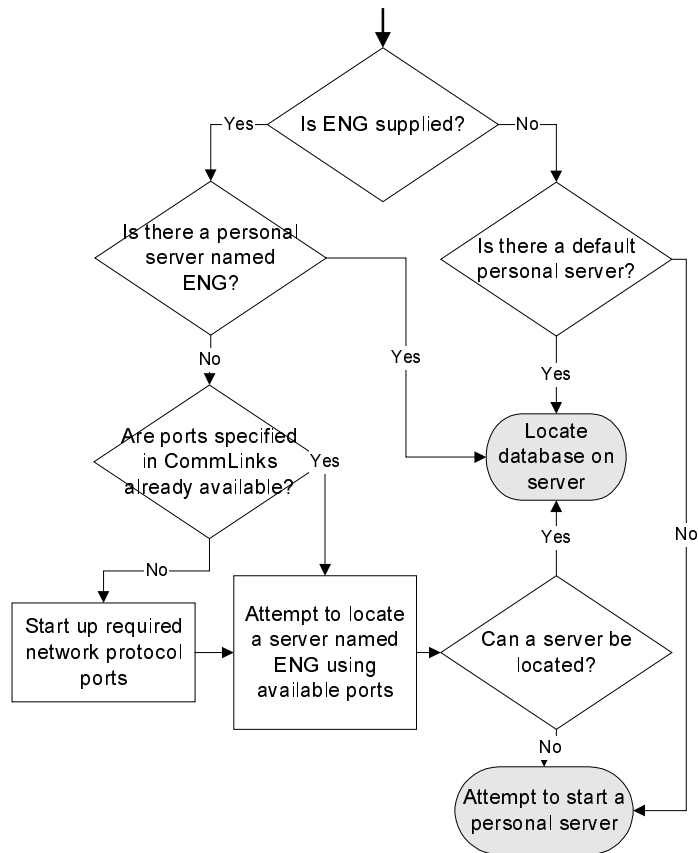
- ◆ **Failure** Failure at this stage occurs only if a data source that does not exist in the client connection file is specified in the connection string or SQLCONNECT.

- ◆ **Common parameters** Depending on other connections already in use, some connection parameters may be ignored. These include the following:
 - ◆ **Autostop** Ignored if the database is already loaded.
 - ◆ **CommLinks** The specifications for a network protocol are ignored if another connection has already set parameters for that protocol.
 - ◆ **CommBufferSize** Ignored if another connection has already set this parameter.
 - ◆ **CommBufferSpace** Ignored if another connection has already set this parameter.
 - ◆ **Unconditional** Ignored if the database is already loaded or the server is already running.

When the list of connection parameters is complete, it is used by the interface library to attempt to connect.

Locating a server

The next step in establishing a connection is to attempt to locate a server. If the connection parameter list includes a server name (ENG parameter), a search is carried out first for a personal server of that name. Following that, a search is carried out over a network. If no ENG parameter is supplied, a default server is looked for.



☞ If a server is located, the next step is to locate or load the required database on that server. For information, see "Locating the database" on page 54.

☞ If no server is located, the next step is to attempt to start a personal server. For information, see "Starting a personal server" on page 55.

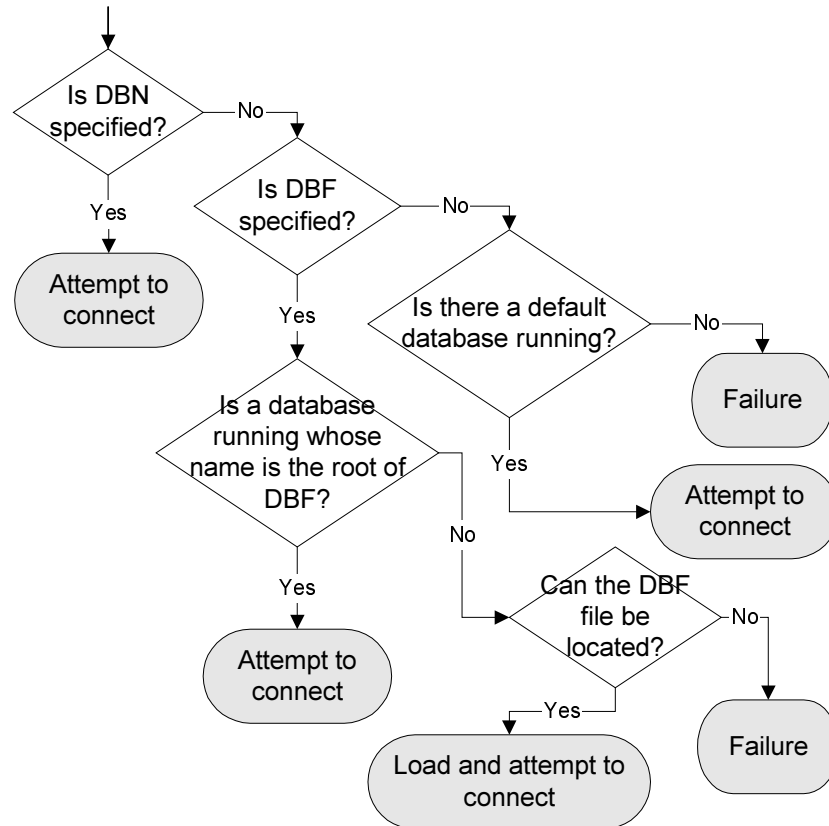
Notes

- ◆ For local connections, locating a server is simple. For connections over a network, you can use the **CommLinks** parameter to tune the search in many ways by supplying network communication parameters.
- ◆ The network search involves a search over one or more of the protocols supported by Adaptive Server Anywhere. For each protocol, the network library starts a single **port**. A single port is used for all connections over that protocol at any one time.

- ◆ A set of network communication parameters can be specified for each network port in the argument to the **CommLinks** parameter. These parameters are used only when the port is first started. If a particular network port is already started, any connection parameters for that port in **CommLinks** are ignored.
- ◆ Each attempt to locate a server (the local attempt and the attempt for each network port) involves two steps. The first is to look in the server name cache to see if a server of that name is available. The second is to use the available connection parameters to attempt a connection.

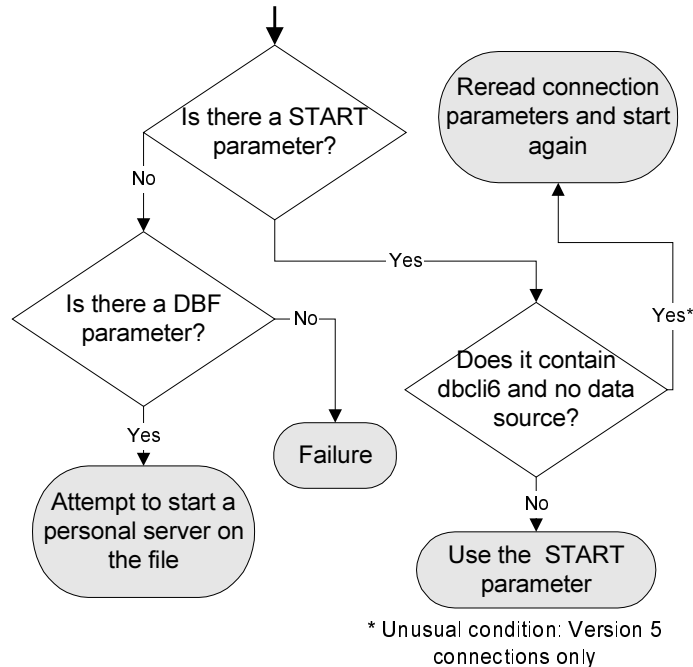
Locating the database

If a server is successfully located, the next step is to locate the database. The following figure illustrates how databases are located:



Starting a personal server

If no server can be located, the interface libraries attempt to start a personal server using other parameters. The **Start** and **DBF** parameters can be used to start a personal server.



The START parameter takes one of two kinds of argument:

- ◆ **Personal server command line** To start a personal server.
- ◆ **Client compatibility command line** For compatibility with Version 5 applications, a client executable command line is accepted.

If a START parameter is not available, there is an attempt to start a personal server on the file indicated by the DBF. If an ENG parameter is supplied in addition to a DBF parameter, it is used as the name of the server.

Server name caching for faster connections

The network libraries look for a database server on a network by broadcasting over the network using the **CommLinks** connection parameter.

Tuning the broadcast

The **CommLinks** parameter takes as argument a string that lists the protocols to use and, optionally for each protocol, a variety of network communication parameters that tune the broadcast.

☞ For a complete listing of network communications parameters, see "Network communications parameters" on page 54 of the book *Adaptive Server Anywhere Reference Manual*.

Caching server information

Broadcasting over large networks to search for a server of a specific name can be time-consuming. To speed up network connections (except for the first connection to a server), when a server is located, the protocol it was found on and its address are saved to a file.

The server information is saved in a file named *asasrv.ini*, in your Adaptive Server Anywhere executable directory. The file contains a set of sections, each of the following form:

```
[Server name]
Link=protocol_name
Address=address_string
```

How the cache is used

When a connection specifies a server name, and a server with that name is not found, the network library looks first in the server name cache to see if the server is known. If there is an entry for the server name, an attempt is made to connect using the link and address in the cache. If the server is located using this method, the connection is much faster, as no broadcast is involved.

If the server is not located using cached information, the connection string information and **CommLinks** parameter are used to search for the server using a broadcast. If the broadcast is successful, the server name entry in the named cache is overwritten.

Cache precedes CommLinks

If a server name is held in the cache, the cache entry is used before the **CommLinks** string.

Interactive SQL connections

The Interactive SQL utility has a different behavior from the default Embedded SQL behavior when a CONNECT statement is issued while already connected to a database. If no database or server is specified in the CONNECT statement, Interactive SQL connects to the current database, rather than to the default database. This behavior is required for database reloading operations.

☞ For an example, see "CONNECT statement" on page 381 of the book *Adaptive Server Anywhere Reference Manual*.

Testing that a server can be found

The *dbping* command-line utility is provided to help in troubleshooting connections. In particular, you can use it to test if a server with a particular name is available on your network.

The *dbping* utility takes a connection string as a command-line option, but by default only those pieces required to locate a server are used. It does not attempt to start a server.

Examples

The following command line tests to see if a server named Waterloo is available over a TCP/IP connection:

```
dbping -c "eng=Waterloo;CommLinks=tcPIP"
```

The following command tests to see if a default server is available on the current machine.

```
dbping
```

☞ For more information on *dbping* options, see "The DBPing utility" on page 78 of the book *Adaptive Server Anywhere Reference Manual*.

Using integrated logins

	<p>The integrated login feature allows you to maintain a single user ID and password for both database connections and operating system and/or network logins. This section describes the integrated login feature.</p>
Operating systems supported	<p>Integrated login capabilities are available for the Windows NT server only. It is possible for Windows 95 clients as well as Windows NT clients to use integrated logins to connect to a network server running on Windows NT.</p>
Benefits of an integrated login	<p>An integrated login is a mapping from one or more Windows NT user profiles to an existing user in a database. A user who has successfully navigated the security for that user profile and logged in to their machine can connect to a database without providing an additional user ID or password.</p> <p>To accomplish this, the database must be enabled to use integrated logins and a mapping must have been granted between the user profile used to log in to the machine and/or network, and a database user.</p> <p>Using an integrated login is more convenient for the user and permits a single security system for database and network security. Its advantages include:</p> <ul style="list-style-type: none">◆ When connecting to a database using an integrated login, the user does not need to enter a user ID or password.◆ If you use an integrated login, the user authentication is done by the operating system, not the database: a single system is used for database security and machine or network security.◆ Multiple user profiles can be mapped to a single database user ID.◆ The name and password used to login to the Windows NT machine do not have to match the database user ID and password.

Caution

Integrated logins offer the convenience of a single security system but there are important security implications which database administrators should be familiar with.

☞ For more information about security and integrated logins, see "Security concerns: unrestricted database access" on page 62.

Using integrated logins

Several steps must be implemented in order to connect successfully via an integrated login.

❖ **To use an integrated login:**

- 1 Enable the integrated login feature in a database by setting the value of the LOGIN_MODE database option to either **Mixed** or **Integrated** (the option is case insensitive), in place of the default value of **Standard**. This step requires DBA authority).
- 2 Create an integrated login mapping between a user profile and an existing database user. This can be done using a SQL statement or a wizard in Sybase Central.
- 3 Connect from a client application in such a way that the integrated login facility is triggered.

Each of these steps is described in the sections below.

Enabling the integrated login feature

The LOGIN_MODE database option determines whether the integrated login feature is enabled. As database options apply only to the database in which they are found, different databases can have a different integrated login setting even if they are loaded and running within the same server.

The LOGIN_MODE database option accepts one of following three values (which are case insensitive).

- ◆ **Standard** This is the default setting, which does not permit integrated logins. An error occurs if an integrated login connection is attempted.
- ◆ **Mixed** With this setting, both integrated logins and standard logins are allowed.
- ◆ **Integrated** With this setting, all logins to the database must be made using integrated logins.

Caution

Setting the LOGIN_MODE database option to Integrated restricts connections to only those users who have been granted an integrated login mapping. Attempting to connect using a user ID and password generates an error. The only exception to this are users with DBA authority (full administrative rights).

Example

The following SQL statement sets the value of the LOGIN_MODE database option to **Mixed**, allowing both standard and integrated login connections:

```
SET OPTION Public.LOGIN_MODE = Mixed
```

Creating an integrated login

User profiles can only be mapped to an existing database user ID. When that database user ID is removed from the database, all integrated login mappings based on that database user ID are automatically removed.

A user profile does not have to exist for it to be mapped to a database user ID. More than one user profile can be mapped to the same user ID.

Only users with DBA authority are able to create or remove an integrated login mapping.

An integrated login mapping is made either using a wizard in Sybase Central or a SQL statement.

❖ To map an integrated login from Sybase Central:

- 1 Connect to a database as a user with DBA authority.
- 2 Open the Integrated Logins folder for the database, and double-click Add Integrated Login. The Integrated Login wizard is displayed.
- 3 On the first page of the wizard, enter the name of the system (computer) user for whom the integrated login is to be created. You can either select a name from the list or enter a name.

Also, select the database user ID this user maps to. The wizard displays the available database users. You must select one of these. You cannot add a new database user ID.

- 4 Follow the remaining instructions in the Wizard.

❖ To map an integrate login using a SQL statement:

- ◆ The following SQL statement allows Window NT users dmelanso and bhay to log in to the database as the user **DBA**, without having to know or provide the DBA user ID or password.

```
GRANT INTEGRATED LOGIN  
TO dmelanso, bhay  
AS USER dba
```

Connecting from a client application

A client application can connect to a database using an integrated login in one of the following ways:

- ◆ Set the INTEGRATED parameter in the list of connection parameters to **yes**.

- ◆ Specify neither a user ID nor a password in the connection string or connection dialog. This method is available only for Embedded SQL applications, including the Adaptive Server Anywhere administration utilities.

If **INTEGRATED=yes** is specified in the connection string, an integrated login is attempted. If the connection attempt fails and the **LOGIN_MODE** database option is set to **Mixed**, the server attempts a standard login.

If an attempt to connect to a database is made without providing a user ID or password, an integrated login is attempted. The attempt succeeds or fails depending on whether the current user profile name matches an integrated login mapping in the database.

Interactive SQL Examples

For example, a connection attempt using the following Interactive SQL statement will succeed, providing the user has logged on with a user profile name that matches a integrated login mapping in a default database of a server:

```
CONNECT USING 'INTEGRATED=yes'
```

The following Interactive SQL statement...

```
CONNECT
```

...can connect to a database if all the following are true:

- ◆ A server is currently running.
- ◆ The default database on the current server is enabled to accept integrated login connections.
- ◆ An integrated login mapping has been created that matches the current user's user profile name.
- ◆ If the user is prompted with a dialog box by the server for more connection information (such as occurs when using the Interactive SQL utility), the user clicks OK *without* providing more information.

Integrated logins via ODBC

A client application connecting to a database via ODBC can use an integrated login by including the **Integrated** parameter among other attributes in its Data Source configuration.

Setting the attribute **Integrated=yes** in an ODBC data source causes database connection attempts using that DSN to attempt an integrated login. If the **LOGIN_MODE** database option is set to **Standard**, the ODBC driver prompts the user for a database user ID and password.

Security concerns: unrestricted database access

The integrated login feature works by using the login control system of Windows NT in place of the Adaptive Server Anywhere security system. Essentially, the user passes through the database security if they can log in to the machine hosting the database, and if other conditions, outlined in "Using integrated logins" on page 58, are met.

If the user successfully logs in to the Windows NT server as "dsmith", they can connect to the database without further proof of identification provided there is either an integrated login mapping or a default integrated login user ID.

When using integrated logins, database administrators should give special consideration to the way Windows NT enforces login security in order to prevent unwanted access to the database.

In particular, be aware that by default a "Guest" user profile is created and enabled when Windows NT Workstation or Server is installed.

Caution

Leaving the user profile Guest enabled can permit unrestricted access to a database being hosted by that server.

If the Guest user profile is enabled and has a blank password, any attempt to log in to the server will be successful. It is not required that a user profile exist on the server, or that the login ID provided have domain login permissions. Literally any user can log in to the server using any login ID and any password: they are logged in by default to the Guest user profile.

This has important implications for connecting to a database with the integrated login feature enabled.

Consider the following scenario, which assumes the Windows NT server hosting a database has a "Guest" user profile that is enabled with a blank password.

- ◆ An integrated login mapping exists between the user **dsmith** and the database user ID **DBA**. When the user **dsmith** connects to the server with her correct login ID and password, she connects to the database as **DBA**, a user with full administrative rights.

But anyone else attempting to connect to the server as "dsmith" will successfully log in to the server regardless of the password they provide because Windows NT will default that connection attempt to the "Guest" user profile. Having successfully logged in to the server using the "dsmith" login ID, the unauthorized user successfully connects to the database as **DBA** using the integrated login mapping.

Disable the Guest user profile for security

The safest integrated login policy is to disable the "Guest" user profile on any Windows NT machine hosting an Adaptive Server Anywhere database. This can be done using the Windows NT User Manager utility.

Setting temporary public options for added security

Setting the value of the LOGIN_MODE option for a given database to **Mixed** or **Integrated** using the following SQL statement permanently enables integrated logins for that database.

```
SET OPTION Public.LOGIN_MODE = Mixed
```

If the database is shut down and restarted, the option value remains the same and integrated logins are still enabled.

Changing the LOGIN_MODE option temporarily will still allow user access via integrated logins. The following statement will change the option value temporarily:

```
SET TEMPORARY OPTION Public.LOGIN_MODE = Mixed
```

If the permanent option value is **Standard**, the database will revert to that value when it is shut down.

Setting temporary public options can be considered an additional security measure for database access since enabling integrated logins means that the database is relying on the security of the operating system on which it is running. If the database is shut down and copied to another machine (such as a user's machine) access to the database reverts to the Adaptive Server Anywhere security model and not the security model of the operating system of the machine where the database has been copied.

☞ For more information on using the SET OPTION statement see "SET OPTION statement" on page 553 of the book *Adaptive Server Anywhere Reference Manual*.

Network aspects of integrated logins

If the database is located on a network server, then one of two conditions must be met for integrated logins to be used:

- ◆ The user profile used for the integrated login connection attempt must exist on both the local machine and the server. As well as having identical user profile names on both machines, the passwords for both user profiles must also be identical.

For example, when the user **jsmith** attempts to connect using an integrated login to a database loaded on network server, identical user profile names and passwords must exist on both the local machine and application server hosting the database. **jsmith** must be permitted to login to both the local machine and the server hosting the network server.

- ◆ If network access is controlled by a Microsoft Domain, the user attempting an integrated login must have domain permissions with the Domain Controller server and be logged in to the network. A user profile on the network server matching the user profile on the local machine is not required.

Creating a default integrated login user

A default integrated login user ID can be created so that connecting via an integrated login will be successful even if no integrated login mapping exists for the user profile currently in use.

For example, if no integrated login mapping exists for the user profile name JSMITH, an integrated login connection attempt will normally fail when JSMITH is the user profile in use.

However, if you create a user ID named **Guest** in a database, an integrated login will successfully map to the **Guest** user ID if no integrated login mapping explicitly identifies the user profile JSMITH.

The default integrated login user permits anyone attempting an integrated login to successfully connect to a database if the database contains a user ID named **Guest**. The permissions and authorities granted to the newly-connected user are determined by the authorities granted to the **Guest** user ID.