SYBASE®
An SAP® Company

Configuration Guide

Open Client™ and Open Server™

15.7

[Microsoft Windows]

DOCUMENT ID: DC35830-01-1570-02

LAST REVISED: June 2012

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

Contents

About This Book

The *Open Client and Open Server Configuration Guide for Microsoft Windows* contains information about configuring your system to run Open Client™ and Open Server™ products.

Audience

This book is written for system administrators. It discusses configuration tasks and topics in terms of system administration rather than application programming.

How to use this book

This book contains these chapters:

- Chapter 1, “Configuration Overview,” provides an overview of the configuration process and configuration requirements.
- Chapter 2, “Basic Configuration for Open Client,” explains how a client application connects to a server and lists the configuration tasks required for connection.
- Chapter 3, “Basic Configuration for Open Server,” explains how an Open Server application listens for client connection requests and lists the configuration tasks required for connection.
- Chapter 4, “Configuring Open Client for Sybase Failover,” describes steps necessary to configure your Open Client applications to connect to the secondary companion during failover.
- Chapter 5, “Using a Directory Service,” explains how applications get connection information from a directory service and lists configuration tasks required for an application to use a directory service.
- Chapter 6, “Using Security Services,” explains how applications use network-based security services and lists configuration tasks required for an application to use security services. This chapter also includes information about both LAN Manager and Kerberos security services.
- Chapter 7, “Using ocscfg,” explains how to use the ocscfg utility to set environment variables and configure drivers.
- Chapter 8, “Using dsedit,” explains how to use the dsedit utility to configure a directory service or the *sql.ini* file.

- Chapter 9, “Troubleshooting with dsedit,” explains how to use the ocscfg utility to test your network connections.
- Appendix A, “Environment Variables,” lists the environment variables that Open Client and Open Server products use and explains how to set environment variables.
- Appendix B, “Configuration Files,” presents an overview of configuration files and describes:
  - *libtcl.cfg*, the driver configuration file
  - *sql.ini* file
  - *ocs.cfg*, the runtime configuration file
- Appendix C, “Localization,” presents an overview of localization files and describes:
  - *locales.dat* file
  - *objectid.dat* file
  - Localized message files
  - Collating sequence files
- Appendix D, “Secure Sockets Layer in Open Client and Open Server,” describes the Secure Sockets Layer (SSL) support for Open Client and Open Server, and summarizes some system configuration tasks that are required to use the SSL protocol.

Related documents

You can see these books for more information:

- The *Open Server and SDK New Features for Windows, Linux, and UNIX*, which describes new features available for Open Server and the Software Developer’s Kit. This document is revised to include new features as they become available.
- The *Open Server Release Bulletin* for your platform contains important last-minute information about Open Server.
- The *Software Developer’s Kit Release Bulletin* for your platform contains important last-minute information about Open Client™ and SDK.
- The *jConnect™ for JDBC™ Release Bulletin* contains important last-minute information about jConnect.
- The *Open Client Client-Library/C Programmers Guide* contains information on how to design and implement Client-Library applications.

- The *Open Client Client-Library/C Reference Manual* contains reference information for Open Client Client-Library™.
- The *Open Server Server-Library/C Reference Manual* contains reference information for Open Server Server-Library.
- The *Open Client and Open Server Common Libraries Reference Manual* contains reference information for CS-Library, which is a collection of utility routines that are useful in both Client-Library and Server-Library applications.
- The *Open Server DB-Library/C Reference Manual* contains reference information for the C version of Open Client DB-Library™.
- The *Open Client and Open Server Programmers Supplement* for your platform contains platform-specific information for programmers using Open Client and Open Server. This document includes information about:
  - Compiling and linking an application
  - The sample programs that are included with Open Client and Open Server
  - Routines that have platform-specific behaviors
- The *Installation and Release Bulletin Sybase® SDK DB-Library Kerberos Authentication Option* contains information about installing and enabling the MIT Kerberos security mechanism to be used on DB-Library. DB-Library only supports network authentication and mutual authentication in the Kerberos security mechanism.
- The *Open Client Client-Library Migration Guide* contains information on how to migrate Open Client™ DB-Library™ applications to Open Client Client-Library
- The *Open Client and Open Server International Developers Guide* provides information about creating internationalized and localized applications.
- The *Open Client Embedded SQL™/C Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with C applications.
- The *Open Client Embedded SQL™/COBOL Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with COBOL applications.

- The *jConnect for JDBC Programmers Reference* describes the jConnect for JDBC product and explains how to access data stored in relational database management systems.
- The *Adaptive Server® Enterprise ADO.NET Data Provider Users Guide* provides information on how to access data in Adaptive Server using any language supported by .NET, such as C#, Visual Basic .NET, C++ with managed extension, and J#.
- The *Adaptive Server Enterprise ODBC Driver by Sybase® Users Guide* for Microsoft Windows and UNIX, provides information on how to access data from Adaptive Server on Microsoft Windows and UNIX platforms, using the Open Database Connectivity (ODBC) Driver.
- The *Adaptive Server Enterprise OLE DB Provider by Sybase Users Guide for Microsoft Windows* provides information on how to access data from Adaptive Server on Microsoft Windows platforms, using the Adaptive Server OLE DB Provider.
- The *Adaptive Server Enterprise Database Driver for Perl Programmers Guide* provides information for Perl developers to connect to an Adaptive Server database and query or change information using a Perl script.
- The *Adaptive Server Enterprise extension module for PHP Programmers Guide* provides information for PHP developers to execute queries against an Adaptive Server database.
- The *Adaptive Server Enterprise extension module for Python Programmers Guide* provides information about Sybase-specific Python interface that can be used to execute queries against an Adaptive Server database.

Other sources of information

Use the Sybase Getting Started CD and the Sybase Product Documentation Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The Sybase Product Documentation Web site is accessible using a standard Web browser. In addition to product documentation, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Documentation Web site, go to Product Documentation at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click Partner Certification Report.

3 In the Partner Certification Report filter select a product, platform, and timeframe and then click Go.

4 Click a Partner Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1 Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

Table 1: Syntax conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in sans serif font.
variable	Variables, or words that stand for values that you fill in, are in *italics*.
{ }	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean choosing one or more of the enclosed items is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you can select only one of the options shown.
,	The comma means you can choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Open Client and Open Server documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the documentation or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Configuration Overview

Welcome to the *Open Client and Open Server Configuration Guide for Microsoft Windows*. Before you read this document, install Open Client or Open Server following the instructions in the *Software Developer's Kit and Open Server Installation Guide for Microsoft Windows*. Open Client is a part of the Software Developer's Kit (SDK).

This chapter gives an overview of the configuration process for Open Client and Open Server.

About Open Client and Open Server

Open Client provides an application programming interface (API) and Net-Library, which allows communications between Adaptive Server Enterprise and Open Server applications, customer applications, third-party products, and other Sybase products.

Open Server provides the tools and interfaces needed to create custom servers. Like Open Client, a programming API and Net-Library enable communications with clients and other servers. In addition, Open Server provides routines that:

- Handle multiple client connections
- Schedule interactions with clients
- Handle error conditions
- Perform other functions required from a server

See the following documents for detailed information about Open Client and Open Server:

- *Open Client Client-Library/C Reference Manual*
- *Open Server Server-Library Reference Manual*
- *Open Client DB-Library/C Reference Manual*

Overview of configuration

Open Client and Open Server software require specific information to function correctly. Configuration is the process of setting up your system to make this information available.

Open Client and Open Server use configuration information to:

- Initialize the Open Client or Open Server application
- Establish a connection with Adaptive Server Enterprise, or an Open Server application

---

Note To ensure that your application has access to the most recent features, set the version to CS_CURRENT_VERSION.

The initialization process

❖ **Initializing an Open Client and Open Server application**

1. Use the SYBASE environment variable to determine the location of the Sybase installation directory.
2. Use the locale-specific POSIX environment variables LC_*, LANG, LC_ALL, and LC_COLLATE and the *locales.dat* file to determine what language, character set, and collating sequence the application uses.
3. Use the *libtcl.cfg* file to load the directory driver and security driver, as required.

The connection process

Clients and servers communicate through a connection. For a client application to connect to a server application, the server application must be listening for the client connection request.

❖ **Making a connection from Open Client**

1 Use the DSQUERY environment variable to determine the name of the target server.

2 Uses the *sql.ini* file or a directory service to obtain the address of the target server.

Note Open Client uses DSQUERY *only* if the Open Client application does not specify the name of the server.

❖ **Listening for a request in Open Server**

1 Use the DSLISTEN environment variable to determine the name of the Open Server application.

2 Use the *sql.ini* file or a directory service to obtain the Open Server application's address.

Note Use DSLISTEN only if the Open Server application does not specify a server during initialization.

Configuration tasks

You must complete some basic configuration tasks for an Open Client and Open Server product to initialize the application and make a connection.

These tasks include:

- Setting environment variables to specify a target's default server and initial localization values. The values of DSQUERY and DSLISTEN are used if Open Client and Open Server applications do not specify a name of a server.
- Verifying that the address of the target server is available.
- Configuring your network driver, if needed.

There are additional tasks if you are:

- Using a directory service
- Using security services
- Using custom localization values in addition to or in place of initial localization values

CHAPTER 2

Basic Configuration for Open Client

This chapter discusses the basic configuration requirements for Open Client.

Note Except where noted, information in this chapter applies to both DB-Library and Client-Library. Specifically, DB-Library does not use environment variables to determine initial localization values and does not examine the *libtcl.cfg* file. However, DB-Library *does* examine the SYBASE and DSQUERY environment variables.

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

Overview of basic configuration

All Open Client applications require the following basic configuration information obtained during initialization and connection:

- Location of the Sybase installation directory
- Locale name
- Localized message and character set files
- Target server name
- Network address of the target server
- Security mechanism to be used

Location of the Sybase installation directory as defined by the SYBASE environment variable.

In a heterogeneous environment that uses applications built for versions 10.x and later, you must explicitly set the SYBASE, SYBASE_OCS, and PATH environment variables at the command prompt, to point to the current version of the Open Client and Open Server installation.

For example, for applications using version 15.5 products installed in *C:\SYBASE*, open a command prompt and set the following environment variables:

```
set SYBASE=C:\SYBASE
set SYBASE_OCS=OCS-15_0
set PATH=%PATH%;%SYBASE%\%SYBASE_OCS%\bin;
%SYBASE%\%SYBASE_OCS%\dll
```

Locale name

Open Client uses the values of the following POSIX environment variables as locale names (does not apply to DB-Library):

- LC_ALL
- LANG, if LC_ALL is not defined

Open Client later uses this value to obtain localization information from the *locales.dat* file. If LC_ALL, LANG, and sLanguage are not defined, Open Client uses “default” as the locale name.

Localized message and character set files

Open Client looks in the *locales.dat* file for an entry whose name matches the locale name determined in the previous step. Then, it loads the localized messages and character set files specified in the *locales.dat* file.

Name of the target server

Open Client obtains the name of the target server from one of the following sources, in the order listed:

1 The client application, which can provide the server name in the call to ct_connect (or dbopen)

2 The DSQUERY environment variable, if the application does not specify the target server

3 The default name SYBASE, if DSQUERY is not set

The network address of the target server

Open Client gets the address(es) of the target server from the directory service or from the *sql.ini* file:

- Directory service – Open Client looks for an entry in the [NT_DIRECTORY] section of *libtcl.cfg* file to determine where to look for the server address information. The setting of the CS_DS_PROVIDER property determines which [NT_DIRECTORY] entry the application searches for or defaults to the first entry of the [NT_DIRECTORY] section.

- *sql.ini* file – if a directory service is not used or if it is used and fails, Open Client searches for the SERVERNAME entry in *sql.ini* that matches the name, and uses the corresponding target address.

  See "The sql.ini file" on page 74 for information about the *sql.ini* file.

  In a heterogeneous environment that uses applications built for versions 10.x and later, you can still maintain a single *sql.ini* file by passing the address file name to each application, for example:

  ```
  isql -Usa -P -Sconnect50 -Ic:\sybase\ini\sql.ini
  ```

Security mechanism to be used

(Does not apply to DB-Library) If the client application requests network-based security services, Open Client looks in the [SECURITY] section of *libtcl.cfg* to determine which security driver to use.

Configuration tasks

To allow your client application to perform the processes listed previously, complete the tasks in the following subsections.

Set environment variables

❖ **Setting environment variables**

1. Set the LC_ALL or LANG environment variable to the desired locale name. The locale name you specify must correspond to an entry in *locales.dat*. If you do not set LC_ALL or LANG, make sure that the "default" entry in *locales.dat* reflects the localization values your applications will use.

   Verify that you have localization files that match the language, character set, and collating sequence specified in the locales file.

2. If your application uses custom localization values, set the LC_ALL, LC_COLLATE, LC_TYPE, LC_MESSAGE, or LC_TIME environment variable to the locale name.

   If you do not know which environment variable your application uses, set all the environment variables to the desired locale name.

3 Set the DSQUERY environment variable to the name of the target server. If the client application names the target server, you do not need to set DSQUERY. If DSQUERY is not set and the application does not name the server, Open Client uses the server name "SYBASE."

See "Setting environment variables" on page 42 for instructions about how to set environment variables using ocscfg.

Configure the drivers

To configure the directory and security drivers, use the ocscfg utility.

See Chapter 7, "Using ocscfg," for information about configuring drivers.

See "The libtcl.cfg and libtcl64.cfg files" on page 68 for reference information about drivers and *libtcl.cfg*.

Configure sql.ini

❖ **Configuring sql.ini**

1 Make an entry for the target server in *sql.ini* using dsedit.

2 Verify that there is an entry in *sql.ini* whose SERVERNAME element corresponds with the value of the DSQUERY environment variable.

See Chapter 8, "Using dsedit," for information about adding information to *sql.ini*. See "The sql.ini file" on page 74 for information about *sql.ini*.

CHAPTER 3

Basic Configuration for Open Server

This chapter describes the basic configuration requirements for Open Server.

About Open Server applications

Open Server applications fall into three functional categories:

- Standalone
- Auxiliary
- Gateway

The configuration of an Open Server application depends on which category it falls into. See the *Open Server Server-Library/C Reference Manual* for more information about the types of Open Server applications.

Overview of basic configuration

All Open Server applications require the following basic configuration information obtained during initialization and connection

- Location of the Sybase installation directory
- Locale name
- Localized message and character set files

- Name of the target server
- Target server's network address

Location of the Sybase installation directory as defined by the SYBASE environment variable

In a heterogeneous environment that uses applications built for versions 10.x and later, you must explicitly set the SYBASE, SYBASE_OCS, and PATH environment variables at the command prompt, to point to the current version of the Open Client and Open Server installation.

For example, for applications using 15.5 products installed in *C:\SYBASE*, open a command prompt and set the following environment variables:

```
set SYBASE=C:\SYBASE
set SYBASE_OCS=OCS-15_0
set PATH=%PATH%;%SYBASE%\%SYBASE_OCS%\bin;
%SYBASE%\%SYBASE_OCS%\dll
```

Locale name

Open Server uses the values of the following POSIX environment variables as locale names:

- LC_ALL
- LANG, if LC_ALL is not defined

Open Server later uses this value to obtain localization information from the *locales.dat* file. If neither environment variable is defined, Open Server uses "default" as the locale name.

Localized message and character set files

Open Server looks in the *locales.dat* file for an entry whose name matches the locale name determined in step 2. Open Server then loads the localized messages and character set files specified in the *locales.dat* file.

Name of the target server

Name of the target server. Open Server obtains the name of the Open Server application from one of the following sources, in the order listed:

1 The Open Server application, which can provide the server name in the call to srv_init

2 The DSLISTEN environment variable, if the application does not specify its name

3 The default name SYBASE, if DSLISTEN is not set

Target server's network address

Target server's network address. Open Server gets the target server's addresses from the directory service or from *sql.ini*:

- Directory service – Open Server looks for an entry in the [NT_DIRECTORY] section of the *libtcl.cfg* file to determine where to look up server address information. The setting of the CS_DS_PROVIDER property determines which [NT_DIRECTORY] entry the application searches for, or defaults to the first entry of the [NT_DIRECTORY] section.
- *sql.ini* file – if a directory service is not used, or if it is used and fails, Open Server searches for the SERVERNAME entry in *sql.ini* that matches the name as determined in step 4 and uses the corresponding target address.

  In a heterogeneous environment that uses applications built for releases 10.0.x and later, you can maintain a single *sql.ini* file by passing the address file name explicitly to each application, for example:

  ```
  isql -Usa -P -Sconnect50 -Ic:\sybase\ini\sql.ini
  ```

When a client requests a connection that uses a network-based security mechanism, Open Server looks up the corresponding security driver in the [SECURITY] section of *libtcl.cfg*.

Configuration tasks

To allow your Open Server application to perform the process described above, complete these tasks:

- Configure sql.ini or Registry
- Set environment variables
- Configure the drivers

Each task is described in the following sections.

Configure sql.ini or Registry

❖ **Configuring *sql.ini***

1. Make an entry for the server's name and directory in *sql.ini* using dsedit.
2. Verify that there is an entry in *sql.ini* whose SERVERNAME element corresponds with the value of the DSLISTEN environment variable.

❖ **Configuring Registry**

1 Open System in Control Panel.

2 Click Environment Variables on the Advanced tab.

3 Edit the following system variables:

   a DSLISTEN to set the value to the name of the Open Server application, as listed in *sql.ini* or directory service.

   b DSQUERY to set the value to the name of the target server, as listed in *sql.ini* or directory service.

4 Click OK in the Environment Variables window to set the new values.

See Chapter 8, "Using dsedit," for instructions about using dsedit.

See "The sql.ini file" on page 74 for reference information about *sql.ini*.

Set environment variables

Set the following environment variables:

- Set the LC_ALL or LANG environment variable to the desired locale name.

  The locale name you specify must correspond to an entry in *locales.dat*. If you do not set LC_ALL or LANG, make sure that the "default" entry in *locales.dat* reflects the localization values your applications will use.

  Make sure you have localization files that match the language, character set, and collating sequence specified in the locales file.

- If your application uses **custom localization values**, set the LC_ALL, LC_COLLATE, LC_TYPE, LC_MESSAGE, or LC_TIME environment variable to the locale name.

  If you do not know which environment variable your application uses, set all the environment variables to the desired locale name.

- Set the DSLISTEN environment variable to the name of the Open Server application.

  If the name of the Open Server application is coded into the application, you do not need to set DSLISTEN. If DSLISTEN is not set and the application does not name the server, Open Server uses the server name SYBASE.

- If the Open Server application acts as a gateway application, set the DSQUERY environment variable to the name of the target server.

See “Setting environment variables” on page 42 for instructions about how to set environment variables using ocscfg.

Configure the drivers

Use the ocscfg utility to configure the network, directory, and security drivers.

See Chapter 7, “Using ocscfg,” for information about configuring drivers.

See “The libtcl.cfg and libtcl64.cfg files” on page 68 for reference information about drivers and *libtcl.cfg*.

CHAPTER 4

Configuring Open Client for Sybase Failover

The Sybase Failover feature is documented in the *Adaptive Server Enterprise Using Sybase Failover in a High Availability System* guide. This chapter describes steps necessary to configure your Open Client applications to connect to the secondary companion during failover, information that is not included in that document.

Note DB-Library does not support HA Failover. Embedded SQL™ (ESQL) for C and COBOL supports HA Failover starting with version 12.5.

Adding a hafailover line to the *sql.ini* file

Clients with the failover property automatically reconnect to the secondary companion when the primary companion crashes or when you issue shutdown or shutdown with nowait, triggering failover. To give a client the failover property, you must add a line labeled "hafailover" to the *sql.ini* file to provide the information necessary for the client to connect to the secondary companion. You can add this line using either a file editor or the dsedit utility.

The following is a *sql.ini* entry for a symmetric configuration between the "MONEY1" and "PERSONNEL1" companions:

```
[MONEY1]
 master=TCP,FN1,9835
 query=TCP,FN1,9835
```

```
 hafailover=PERSONNEL1


[PERSONNEL1]
 master=TCP,HUM1,7586
 query=TCP,HUM1,7586
 hafailover=MONEY1
```

For more information about adding this information to the *sql.ini* file, see "Configure sql.ini" on page 8.

Note Client applications must resend any queries that were interrupted by failover. Other information specific to the connection, such as cursor declarations, also need to be restored.

Client-Library application changes

Note An application installed in a cluster must be able to run on both the primary and secondary companions. If you install an application that requires a parallel configuration, the secondary companion must also be configured for parallel processing so it can run the application during failover.

You must modify any application written with Client-Library calls before it can work with Sybase's Failover software. The following steps describe the modifications:

1 Set the CS_HAFAILOVER property using the ct_config and ct_con_props Client-Library API calls. Legal values for the property are CS_TRUE and CS_FALSE. The default value is CS_FALSE. You can set this property at either the context or the connection level. The following is an example of setting the property at the context level:

```
CS_BOOL bhafailover = CS_TRUE;
retcode = ct_config(context, CS_SET, CS_HAFAILOVER,
&bhafailover, CS_UNUSED, NULL);
```

The following shows the property set at the connection level:

```
CS_BOOL bhafailover = CS_FALSE;
retcode = ct_con_props(connection, CS_SET,
```

```
CS_HAFAILOVER, &bhafailover, CS_UNUSED, NULL);
```

2 Handle failover messages. As soon as the companion begins to go down, clients receive an informational message that failover is about to occur. Treat this as an informational message in the client error handlers.

3 Confirm failover configuration. Once you have set the failover property and the *sql.ini* file has a valid entry for the secondary companion server, the connection becomes a failover connection, and the client reconnects appropriately.

   However, if the failover property is set but the *sql.ini* file does not have an entry for the hafailover server (or vice-versa), it does not become a failover connection. Instead, it is a normal non-high availability connection with the failover property turned off. You must check the failover property to know whether or not the connection is a failover connection. You can do this by calling ct_con_props with an *action* of CS_GET.

4 Check return codes. When a successful failover occurs, calls to ct_results and ct_send return CS_RET_HAFAILOVER depending on the type of connection:

   - On a synchronous connection, the API call returns CS_RET_HAFAILOVER directly.
   - On an asynchronous connection, the API returns CS_PENDING and the callback function returns CS_RET_HAFAILOVER.

   Depending on the return code, the application can do the required processing, such as sending the next command to be executed.

5 Restore option values. Any set options that you have configured for this client connection (for example, set role) were lost when the client disconnected from the primary companion. Reset these options in the failed-over connection.

6 Rebuild your applications, linking them with the libraries included with the failover software.

Note You cannot connect clients with the failover property (for example, isql -Q) until you issue sp_companion resume. If you do try to reconnect them after issuing sp_companion prepare_failback, the client hangs until you issue sp_companion resume.

Using isql with Sybase Failover

To use isql to connect to a primary server with failover capability, you must:

- Choose a primary server that has a secondary companion server specified in its *sql.ini* file entry.
- Use the -Q command-line option.

If your *sql.ini* file contained the example entry given in "Adding a hafailover line to the sql.ini file," you could use isql with failover by entering:

```
isql -S PERSONNEL1 -Q
```

CHAPTER 5

Using a Directory Service

Client-Library and Server-Library applications can use directory services to keep track of information about servers. This chapter describes how a directory service works and how to configure one.

Note DB-Library supports only LDAP directory service.

Overview of directory services

A directory service manages the creation, modification, and retrieval of information about network entities. As an alternative to *sql.ini*, Client-Library and Server-Library applications use a directory service to obtain information about servers.

The advantage of using a directory service is that you do not need to update multiple *sql.ini* files when a new server is added to your network or when a server moves to a new address.

Different platforms use different directory service providers; in Microsoft Windows, you can use Windows Registry and LDAP.

LDAP

Lightweight Directory Access Protocol (LDAP) is used to access directory listings. A directory listing, or service, provides a directory of names, profile information, and machine addresses for every user and resource on the network. It can be used to manage user accounts and network permissions.

LDAP servers are typically hierarchical in design and provide fast lookups of resources. LDAP can be used as a replacement to the traditional Sybase *sql.ini* file to store and retrieve information about Sybase servers.

Any type of LDAP service, whether it is an actual server or a gateway to other LDAP services, is called an LDAP server. An LDAP driver calls LDAP client libraries to establish connections to an LDAP server. The LDAP driver and client libraries define the communication protocol, such as whether encryption is enabled, and the contents of messages exchanged between clients and servers. Messages are operators, such as client requests for read, write, and queries, and server responses, including data-format information.

LDAP directory services versus the Sybase *sql.ini* file

LDAP directory services are a convenient alternative to the typical Sybase *sql.ini* file, which stores server information in a "flat" file. As a result, any changes to server information in the *sql.ini* file need to be updated on each machine (client and server) in the enterprise.

With LDAP directory services, the integration of user, resource, and security information in a centralized repository makes administration of resource information much easier. In addition, LDAP services provide:

- A single, hierarchical view of information, such as users, software, resources, networks, files
- A single sign-on for servers and distributed enterprise applications
- User login and role information for access control to sensitive data

User roles can be assigned to a single individual, such as the system administrator, or to large groups of users, such as accounting department personnel. Roles determine what information and servers users can access, and what, if any, read and write permission they possess. Multiple users with the same user role can be multiplexed to a few server connections, saving resources and increasing scalability.

Table 5-1 highlights the differences between the Sybase *sql.ini* file and LDAP server:

Table 5-1: sql.ini file versus LDAP directory services

The *sql.ini* file	Directory services
Platform-specific	Platform-independent
Specific to each Sybase installation	Centralized and hierarchical
Contains separate master and query entries	Contains one entry for each server that is accessed by both clients and servers
Cannot store metadata about the server	Stores metadata about the server

The traditional *sql.ini* file specifying a TCP connection and a failover machine looks like this:

```
[MONEY]
master=TCP, huey, 5000
query=TCP, huey, 5000
hafailover=PERSONEL
[PERSONEL]
master=TCP, huey, 5000
query=TCP, huey, 5000
hafailover=MONEY
```

An example of an LDAP entry with TCP and a failover machine looks like this:

```
dn: sybaseServername=foobar, dc=sybase,dc=com
objectClass: sybaseServer
sybaseVersion: 15501
sybaseServername: foobar
sybaseService: ASE
sybaseStatus: 4
sybaseAddress: TCP#1#foobar 5000
sybaseRetryCount: 12
sybaseRetryDelay: 30
sybaseHAServernam: secondary
```

All entries in the LDAP directory service are called entities. Each entity has a distinguished name (DN) and is stored in a hierarchical tree structure based on its DN. This tree is called the directory information tree (DIT). Client connections specify where to begin the search of an LDAP server by specifying a DIT base during connection. Table 5-2 lists valid DIT-base values.

Table 5-2: Sybase LDAP entry definitions

Attribute name	Value type	Description
sybaseVersion	Integer	Server version number.
sybaseServername	Character string	Server name.
sybaseService	Character string	Service type: Sybase Adaptive Server.

Attribute name	Value type	Description
sybaseStatus	Integer	Status: 1 = Active, 2 = Stopped, 3 = Failed, 4 = Unknown.
sybaseAddress	String	Each entry in the address string is separated by the # character. Each server address includes: • Protocol: TCP, NAMEDPIPE. • The value of the sybaseStatus. • Address: any valid address for the protocol type.
sybaseSecurity (optional)	String	Security OID (object ID).
sybaseRetryCount	Integer	This attribute is mapped to CS_RETRY_COUNT, which specifies the number of times that ct_connect retries the sequence of network addresses associated with a server name.
sybaseRetryDelay	Integer	This attribute is mapped to CS_LOOP_DELAY, which specifies the delay, in seconds, that ct_connect waits before retrying the entire sequence of addresses.
sybaseHAservername (optional)	String	A secondary server for failover protection.

Sybase provides LDAP directory schema in *%SYBASE%\%SYBASE_OCS%\ini* for the following LDAP services:

- *sybase.schema* - it contains the directory schema to be used with OpenLDAP servers.
- *sybase-schema.conf* - it contains the directory schema but in a Netscape-specific syntax.
- *sybase.ldf* - it contains directory schema in Unicode format for a Microsoft Active Directory.

In the previous example, the entity describes an Adaptive Server named “foobar” listening on a TCP connection with a port number of 5000. This entity also specifies a retry count of 12 (times) and a retry delay of 30 (seconds). sybaseRetryCount and sybaseRetryDelay map to CS_RETRY_COUNT and CS_LOOP_DELAY, respectively. When Client-Library finds an address where a server responds, the login dialog begins between Client-Library and the server. Client-Library does not retry any other addresses if the login attempt fails.

The most important entity is the address attribute, which contains the information for how to set up a connection to the server and how the server listens for incoming connections. For entries to be usable by different Sybase products on different platforms, the Protocol field and the Address field in an address attribute (for example, “TCP” and “foobar 5000”) should be in a platform and product independent form.

Since LDAP supports multiple entries for each attribute, each address attribute must contain the address of a single server, including protocol, access type, and address. See sybaseAddress in Table 5-2 on page 21.

The following example is an LDAP entry for a Microsoft Windows server listening on two addresses, with different connection protocols:

```
sybaseAddress = TCP#1#TOEJAM 4444
sybaseAddress = NAMEPIPE#1#\pipe\sybase\query
```

Each entry in the address field is separated by the # character. Table 5-2 on page 21 defines the values for each field in the address attribute.

Server objects and attributes

The directory service must contain information about servers accessed by your Open Client.

A directory service identifies a server entry as a directory object. Each directory object has a unique set of attributes. You can create, view, and modify server object entries with dsedit. Refer to Chapter 8, “Using dsedit,” for more information.

Directory drivers

Open Client and Open Server software uses a directory driver to retrieve information from a directory service.

A directory driver is a dynamically-linked Sybase library that provides Open Client and Open Server software with a generic interface to a specific directory service. Sybase provides a directory driver for each supported directory service.

Directory drivers are listed in the *libtcl.cfg* file. See “The libtcl.cfg and libtcl64.cfg files” on page 68 for reference information about directory drivers and *libtcl.cfg*.

How applications use a directory service

Client-Library, Server-Library, and DB-Library determine whether to use a directory service or *sql.ini* as follows:

1 If the Client-Library or the Server-Library application specifies a directory driver, Client-Library by calling ct_con_props (CS_SET, CS_DS_PROVIDER) and Server-Library by calling srv_props (CS_SET, SRV_DS_PROVIDER), the application checks in the [DIRECTORY] section of *libtcl.cfg* for a matching driver and loads that driver.

See "The libtcl.cfg and libtcl64.cfg files" on page 68 for reference information about directory drivers and *libtcl*.cfg*.

Note Step 1 does not apply to DB-Library application. Use steps 2 and 3 for specifying directory service for DB-Library.

2 If the client application does not specify a directory driver, Client-Library, Server-Library, and DB-Library loads the directory driver listed by the first entry in the [DIRECTORY] section of *libtcl.cfg*.

3 Client-Library, Server-Library, and DB-Library fall back and use *sql.ini* to obtain the server's address if any of the following are true:

- *libtcl.cfg* does not exist.
- There are no entries in the [DIRECTORY] section of *libtcl.cfg*.
- The specified directory driver fails to load.
- *libtcl*.cfg* is overridden at the context level when the CS_IFILE property is set with ct_config.

You use the *libtcl*.cfg* file to specify the LDAP server name, port number, DIT base, user name, and password to authenticate the connection to an LDAP server.

What you should know about the *libtcl*.cfg* file:

- Values specified in the *libtcl*.cfg* file serve as the defaults for the CS_* property, which is set with the ct_con_props routine. You can override these values by explicitly setting the ct_con_props routine for that specific connection.
- If you do not specify either the password or the user name in the *libtcl*.cfg* file, the connection is anonymous.

- If the password begins with an "0x," the connection properties assume that the password is encrypted. See "Encrypting the password" on page 71.
- On 64-bit platforms, Open Client and Open Server contain both 32-bit and 64-bit binaries. You should edit both the *libtcl.cfg* and the *libtcl64.cfg* files to ensure compatibility between 32-bit and 64-bit applications.

The *libtcl*.cfg* file is located in *%SYBASE%\%SYBASE_OCS%\ini*.

How applications use LDAP directory services

To use Sybase LDAP features, you must install and configure an LDAP server according to the vendor-supplied instructions. Sybase does not provide the LDAP server. Sybase provides Netscape LDAP SDK client libraries and Sybase Open Client and Open Server includes an LDAP driver, located in *%SYBASE%\%SYBASE_OCS%\dll*.

The Netscape LDAP SDK library is located in *%SYBASE%\%SYBASE_OCS%\lib3p* and the environment variable is PATH.

When the LDAP driver connects to the LDAP server, the server establishes the connection based on two authentication methods—anonymous access, and user name and password authentication.

- Anonymous access – does not require any authentication information; therefore, you do not have to set any properties. Anonymous access is typically used for read-only privileges.
- User name and password – can be specified in the *libtcl.cfg* file *(libtcl64.cfg* file for 64-bit platforms) as an extension to the LDAP URL (see "The libtcl.cfg and libtcl64.cfg files" on page 68) or set with property calls to Client-Library. The user name and password that are passed to the LDAP server, using ctlib, are separate and distinct from the user name and password used to log in to Adaptive Server. Sybase strongly recommends that you use user name and password authentication.

Authentication

A client application creates a connection to an LDAP server using the host name and port number or IP address. This connection is called a "bind" and can be unsecured or have user name and password authentication. The type of access allowed is determined by the server.

Anonymous connections

A connection in which authentication is not required is called an anonymous connection. LDAP and Netscape Directory Services default to allow anonymous connections.

Anonymous access:

- Does not require any authentication information, such as a password, to establish a connection.
- Does not require that any additional properties be set to make a connection.
- Is generally read access only.

User name and password authentication

For access permissions that allow write capabilities, Sybase recommends the use of basic security. User names and passwords can provide a basic level of security for a connection to the LDAP server. You can store user names and passwords in the *libtcl.cfg* file on 32-bit platforms and *libtcl64.cfg* file on 64-bit platforms, or set them with Client-Library properties.

See Appendix B, "Configuration Files," for information about the *libtcl*.cfg* files and encrypting passwords in the configuration file.

Enabling LDAP directory services

Note LDAP is only supported with reentrant libraries. You must use isql_r, instead of isql, when connecting to a server using LDAP directory services.

❖ **Setting up to use a directory service**

1. Configure the LDAP server according to the vendor-supplied documentation.
2. Add the LDAP library directory to your path for your platform. For example:

   ```
   PATH=%PATH%:%SYBASE%\%SYBASE_OCS%\lib3p
   ```

3. Configure the *libtcl*.cfg* file to use directory services. Use any standard ASCII text editor to:
   - Remove the semicolon (;) comment markers from the beginning of the LDAP URL lines in the *libtcl*.cfg* file under the [DIRECTORY] entry.

- Add the LDAP URL under the [DIRECTORY] entry. See Table 5-2 for supported LDAP URL values.

Note The LDAP URL must be on a single line.

```
ldap=libsybdldap.dll ldap://host:port/ditbase??scope????
     bindname=username password
```

For example:

```
[DIRECTORY]
ldap=libsybdldap.dll ldap://huey:11389/dc=sybase,dc=com??
     one????bindname=cn=Manager,dc=sybase,dc=com secret
```

"*one*" indicates the scope of a search that retrieves entries one level below the DIT base. Table 5-3 defines the keywords for the *ldapurl* variables.

Table 5-3: ldapurl variables

Keyword	Description	Default	CS_* property
host (required)	The host name or IP address of the machine running the LDAP server	None	
port	The port number on which the LDAP server is listening	389	
ditbase (required)	The default DIT base	None	CS_DS_DITBASE
username	Distinguished name (DN) of the user to authenticate	NULL (anonymous authentication)	CS_DS_PRINCIPAL
password	Password of the user to be authenticated	NULL (anonymous authentication)	CS_DS_PASSWORD

4 Verify that the environment variable points toPATH*%SYBASE%\%SYBASE_OCS%\lib3p*.

5 Add your server entry to the LDAP server using dsedit. See "Making and modifying server entries" on page 52 and "Adding a server to the directory services" on page 51.

Multiple directory services with LDAP

You can specify multiple directory services for high-availability failover protection. Not every directory service in the list needs to be an LDAP server, for example:

```
[NT_DIRECTORY]
   ldap=libsybdldap.dll ldap://test:389/dc=sybase,dc=com
   dce=libddce.dll ditbase=/.:/subsys/sybase/dataservers
   ldap=libsybdldap.dll ldap://huey:11389/dc=sybase,dc=com
```

In this example, if the connection to `test:389` fails, the connection fails over to the DCE driver with the specified DIT base. If this also fails, a connection to the LDAP server on `huey:11389` is attempted. Different vendors employ different DIT-base formats. See the *Open Client Client-Library/C Reference Manual*.

Importing Microsoft Active Directory schema

You can import *sybase.ldf* into the Active Directory (AD) or into an Active Directory Application Mode (ADAM) instance using the ldifde.exe command provided in the ADAM installation. To import the directory schema, run ldifde.exe from the ADAM installation using this syntax:

```
ldifde -i -u -f sybase.ldf -s server:port -b username
domain password -j . -c "cn=Configuration,dc=X"
#configurationNamingContext
```

Creating a container for Sybase server entries

After you have successfully imported the schema into the Active Directory, create a container for the Sybase server entries and set appropriate read and write permissions for the container and its child objects.

For example, A container with a relative distinguished name (RDN) “CN=SybaseServers” is created in the root of the Active Directory for domain “mycompany.com” to store and retrieve Sybase server entries. The root distinguished name (rootDN) for this container is reflected in the *libtcl.cfg* file as:

```
ldap=libsybdldap.dll ldap://localhost:389/
cn=SybaseServers,dc=mycompany,dc=com??...
```

If you create a dedicated user account name “Manager” with password “secret” in the Active Directory to add and modify Sybase server entries, the complete entry in the *libtcl.cfg* file is:

```
ldap=libsybdldap.dll
ldap://localhost:389/cn=SybaseServers,dc=mycompany,
dc=com????bindname=cn=Manager,cn=Users,dc=mycompay,
dc=com?secret
```

After setting the appropriate read and write permissions, you can use Sybase utility programs such as dscp or dsedit to store, view, and modify Sybase server entries in the Active Directory.

Note For more information about extending an Active Directory schema, search for “Extending the Schema” on the Microsoft Web site.

Connecting to LDAP using SSL/TLS

You can set up a secure connection to an LDAP directory server using SSL or TLS on all supported platforms. To establish a secure connection between a client and an LDAP Directory Server, use either of the following methods:

- Establish a secure connection to the secure port of the LDAP server (typically port number 636) by entering the following syntax in the *libtcl.cfg* file:

  ```
  [NT_DIRECTORY]
  ldap=libsybdldap.dll
  ldaps:// huey:636/dc=sybase,dc=com????
  bindname=cn=Manager,dc=Sybase,dc=com?secret
  ```

  If no port number is specified with ldaps://, port number 636 is used by default.

- Upgrade a normal connection (typically port number 389 of the LDAP Server) to a secure one, using StartTLS. To upgrade the connection, enter this syntax in the *libtcl.cfg* file:

  ```
  [NT_DIRECTORY]
  ldap=libsybdldap.dll starttls
  ldap:// huey:389/dc=sybase,dc=com????
  bindname=cn=Manager,dc=Sybase,dc=com?secret
  ```

If no port number is specified with ldap://, port number 389 is used by default.

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

CHAPTER 6

Using Security Services

Client-Library and Server-Library applications can use the security services provided by third-party security software to authenticate users and protect data transmitted between machines on a network.

This chapter discusses how network-based security works and what you need to configure to use it.

Overview of network-based security

In a distributed client/server computing environment, intruders can view or tamper with confidential data. To counteract this possibility, network-based security takes advantage of third-party distributed security software to authenticate users and protect data transmitted between machines on a network.

Security mechanisms

Sybase defines a security mechanism as external software that provides security services for a connection. Different platforms can use different security mechanisms.

Both Microsoft Windows NT LAN Manager (SSPI) and Kerberos provide security services for servers and clients on Microsoft Windows.

You can specify the security mechanisms that a server supports in *sql.ini* or a directory service:

- The optional secmech line in *sql.ini* entry specifies the security mechanisms that a server supports.

- The optional secmech attribute in a directory service entry describes the security mechanisms that a server supports.

When a client gets the server's address, it can verify that the server supports the security mechanism that the client is using:

- If there is a secmech line or attribute and security mechanisms are listed, then only those security mechanisms are allowed.
- If there is no secmech line or attribute, then all security mechanisms are allowed.
- If there is a secmech line or attribute, but no security mechanisms are listed, then the server does not support any security mechanisms.

Security drivers

Sybase provides security drivers that allow Client-Library and Server-Library to communicate with the security mechanism. Each Sybase security driver maps a generic interface to the security provider's interface.

To use a security mechanism on a connection, both items below must be true:

- The client and server must use compatible security drivers. For example, a client using a Microsoft Windows LAN Manager driver requires a server using a Microsoft Windows LAN Manager driver.
- The client application must request services by setting connection properties before connecting to the server.

Security services

Each security mechanism provides a set of security services that establish a secure connection between a client and a server. Each security service addresses a particular security concern. Security services can be divided into two broad categories:

- Authentication services
- Per-packet security services

See the *Open Client Client-Library/C Reference Manual* for a complete discussion of security services.

Client-Library applications set connection properties to request a mechanism's services. Open Server applications read the properties of a client thread to determine which services are being performed.

LAN Manager security services

Windows LAN Manager services provide the following:

- Network authentication based on LAN Manager user namespace
- Data integrity
- Replay detection
- Out-of-sequence detection

See the following section for information about Kerberos security features.

Kerberos security services

The Kerberos security mechanism provides the following services:

- Network authentication
- Mutual authentication
- Data integrity
- Data confidentiality
- Replay detection
- Out-of-sequence detection
- Credential delegation

For a description of these security services, see the *Open Client Client-Library/C Reference Manual*. See "Client-Library and security services" on page 38 for an overview of how client applications use security services.

Configuring CyberSafe Kerberos

The following considerations apply specifically to client applications that use CyberSafe Kerberos security services:

- Install the CyberSafe Kerberos software on your system for Open Client and Open Server 12.5 or later.

- The *gssapi32.dll* file must be in the library path while running your Client-Library application. Sybase does not provide this DLL, but it is included with some CyberSafe Kerberos products. If this DLL is not included with your CyberSafe Kerberos product, contact CyberSafe Kerberos to obtain their *GSS-API* library.
- Configure the security section of the *libtcl.cfg* configuration file.
- Set the desired security features using ct_con_props. If you want to use the default credentials, do not set any credential properties.
- Verify that the application has a preexisting user credential to connect to the server. In other words, the user of the application must log in to CyberSafe Kerberos before running the client application. To do so, use the single sign-on feature or the CyberSafe kinit utility.
- If a user name is supplied, it must match the user's preexisting credential. If a user name is not supplied, Client-Library connects to the server using the user name associated with the user's CyberSafe Kerberos credential.
- The following environment variables set the paths to the credentials cache file, configuration file, and realms file. If the corresponding file is located in a non-default directory, set the environment variable to the file's full path:
  - CSFC5CCNAME – credentials cache file
  - CSFC5CONFIG – configuration file
  - CSFC5REALMS – realms file

  For more information, refer to your CyberSafe Kerberos documentation.
- No extra flags are required when compiling your Client-Library applications to use CyberSafe Kerberos security services.
- Once you have configured Open Client and Open Server and CyberSafe Kerberos, use the following command (without -U and -P arguments) to test your configuration:

```
isql -V
```

Note Some tasks described here require you to use the CyberSafe Kerberos administration tools. See your CyberSafe Kerberos documentation for information.

Configuring MIT Kerberos

- Install and configure the MIT software on your system, version 2.6.5 or later.
- Configure the security section of the *libtcl.cfg* configuration file.
- Set the desired security features using ct_con_props, or use the default credentials by not setting credential properties.
- Verify that the application has a preexisting user credential to connect to the server. In other words, the user of the application must log in to the Kerberos environment using the kinit utility, before running the client application.
- If a user name is supplied, it must match the user's preexisting credential. If a user name is not supplied, Client-Library connects to the server using the user name associated with the user's credential.
- The environment variable KRB5CCNAME sets the path to the credentials cache file. If the corresponding file is located in a non-default directory, set the environment variable to the file's full path.

  For more information, refer to your documentation.
- The MIT GSS library, *gssapi32.dll*, must be specified in the *libtcl.cfg* file using the libgss keyword. Sybase recommends providing the full path to the Kerberos driver.
- No extra flags are required when compiling your Client-Library applications to use Kerberos security services.
- Once you have configured Open Client and Open Server and Kerberos, you can use isql to test your configuration.

Credential delegation for MIT Kerberos

The Kerberos security driver supports credential delegation when using the MIT Kerberos Generic Security Services (GSS) library. This allows you to set up an Open Server gateway application that uses the delegated client credentials when establishing a connection with a remote server.

❖ **Establishing a connection with a remote server using credential delegation**

This is an example of a call sequence you can employ when using credential delegation. The ctos example in *$SYBASE/OCS-15_0/sample/srvlibrary/connect.c* contains an example of the properties mentioned here:

1 The client application requests for credential delegation and forwards the credential to the gateway connection using:

```
ct_con_props(..., CS_SET, SRV_SEC_DELEGATION, ...)
```

2 The connection handler of the gateway application checks whether the client requested credential delegation:

```
if (srv_thread_props(..., CS_GET,
     SRV_T_SEC_DELEGATION, ...))
    {...}
```

3 The connection handler retrieves the delegated client credentials:

```
srv_thread_props(..., CS_GET,
     SRV_T_SEC_DELEGCRED, ...)
```

4 The client application sets the delegated credentials in the Client-Library connection structure for use in connecting to the remote server:

```
ct_con_props(..., CS_SET, CS_SEC_CREDENTIALS, ...)
```

5 The client application attempts to connect to the remote server using ct_connect.

You can also request for credential delegation using the isql and bcp option -Vd. For more information, see the *Open Client and Open Server Programmers Supplement for Microsoft Windows*.

For detailed information on using credential delegation, see the *Open Server Server-Library/C Reference Manual* and the *Open Client Client-Library/C Reference Manual*.

Using Windows Security SSPI

If Kerberos support is provided by the Microsoft Windows Security Support Provider Interface (SSPI), edit the *csfkrb5* entry in the *libtcl.cfg* file to specify the *libsspiwrapper.dll* as the GSS library.

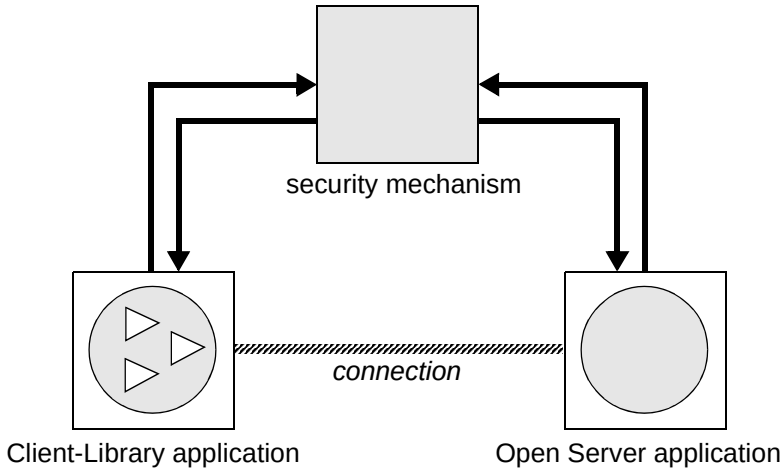
For example:

```
 csfkrb5=LIBSKRB secbase=@REALM libgss=C:\sybase\OCS-
15_0\lib3p\libsspiwrapper.dll
```

How applications use security services

Client-Library and Server-Library applications can use a security mechanism to perform authentication and per-packet security services. The security mechanism behaves like a clearinghouse through which Client-Library and Server-Library validate information. Figure 6-1 applies to both authentication and per-packet security services.

Figure 6-1: Open Client and Open Server applications using a security mechanism



If an Open Client application requests authentication services, the following process occurs:

1 Client-Library validates the login with the security mechanism. The security mechanism returns a login record, or token. The security mechanism creates the login token based on which security services are requested.

2 Client-Library establishes a transport connection with the Open Server application and sends its login token.

3 Server-Library authenticates the client's login token with the security mechanism. If the login is valid, the Open Server application establishes a secure connection.

If an Open Client application requests per-packet security services, the following process occurs:

1 Client-Library uses the security mechanism to prepare the data packet to send to the Open Server application. Depending on which security services are requested, the security mechanism might encrypt the data or create a cryptographic signature associated with the data.

2 Client-Library sends the data packet to the Open Server application.

3 When Open Server receives the data packet, it uses the security mechanism to perform any required decryption and validation.

See "Security Features" in the *Open Client Client-Library/C Reference Manual* for a detailed explanation of the Client-Library's security features.

Client-Library and security services

You can set connection properties in Open Client applications to request a security mechanism's services. Client-Library determines which security mechanism and services to use on the connection as follows:

1 If the client application names a security driver, Client-Library checks in *libtcl.cfg* for a matching driver and loads that driver.

2 If the client application does not name a security driver, Client-Library loads the first security driver listed in *libtcl.cfg*.

3 If *libtcl.cfg* does not list a security driver, the server authenticates the user if the user supplies the correct password.

Server-Library and security services

Open Server applications can read the properties of a client connection request to determine which security mechanism to use and which services to perform.

By default, an Open Server application supports the security mechanisms listed in *libtcl.cfg*. Administrators can further restrict the list of supported mechanisms by adding a secmech attribute to the server's directory entry or a secmech line to the Open Server application's *sql.ini* file entry.

When an Open Client application requests a security session from an Open Server application:

1 Server-Library reads the security token that was sent with the client connection request. The security token contains the object identifier for the security mechanism that the client uses.

2 If the Open Server application's *sql.ini* entry or directory service entry lists the secmech line/attribute, Server-Library searches the secmech line/attribute for a value corresponding to the object identifier specified in the security token. If a matching value is not found, the connection request is rejected.

3 Server-Library searches *objectid.dat* to match the object identifier with the local name of the security mechanism.

See "The objectid.dat file" on page 87 for reference information about *objectid.dat*.

4 Server-Library loads the security driver associated with the local name of the security mechanism. The security driver is listed in *libtcl.cfg*.

Configuration tasks

To allow your Open Client and Open Server application to use a security service, you must configure a security driver using the ocscfg utility. See Chapter 7, "Using ocscfg," for information about configuring a security driver, and "The libtcl.cfg and libtcl64.cfg files" on page 68 for reference information about security drivers and the *libtcl.cfg* file.

Optionally, to restrict the security mechanisms that a server supports, do one of the following:

- If your application uses the *sql.ini* file, use the dsedit utility to add a secmech line in the server's *sql.ini* file entry.
- If your application uses a directory service, use the dsedit utility to add the secmech attribute to the server's directory service entry.

See Chapter 8, "Using dsedit," for information about adding information to a directory service or the *sql.ini* file.

CHAPTER 7

Using *ocscfg*

This chapter explains how to use the ocscfg utility that allows you to configure your local machine.

Note You can also access dsedit while configuring directory services. See Chapter 8, “Using dsedit,” for instructions for using dsedit.

About ocscfg

You can set three types of configuration information with ocscfg:

- Environment variables
- Directory drivers
- Security drivers

Starting ocscfg

You can start ocscfg from Program Manager, the DOS prompt, or the File Manager. You can also start ocscfg from the Start menu or Windows Explorer.

- To start ocscfg from Program Manager, double-click its icon in the Sybase program group.
- To start ocscfg from the prompt, enter:

  ```
  ocscfg
  ```

- To start ocscfg from the File Manager:
  a Go to *%SYBASE%\%SYBASE_OCS%\bin*, where *%SYBASE%* is the installation directory.
  b Double-click the *ocscfg.exe* file.
- To start ocscfg from the Start menu, choose Start | Programs | Sybase | ocscfg.

Use the top row of tabs to select the configuration function you want to perform. The following table describes the function associated with each tab:

Tab	Function
Environment	Set Sybase-related environment variables. ocscfg defaults to this dialog box at start-up.
Directory Service	Configure directory drivers listed in *libtcl.cfg*. Connect to dsedit.
Security Service	Configure security drivers listed in *libtcl.cfg*.

Setting environment variables

Click the Environment tab to activate the dialog box for setting environment variables.

Setting the SYBASE environment variables

To set the SYBASE environment variable, do one of the following:

- Enter the location of the Sybase installation directory in the SYBASE field.

- Click Browse to view your local directory structure or a remote directory structure. Double-click the appropriate directory to select it.

Note ocscfg uses the SYBASE environment variable to locate the *libtcl.cfg* file. If the SYBASE environment variable is not set correctly, ocscfg cannot locate *libtcl.cfg*.

You can also use the CS_LIBTCL_CFG property to set the path to libtcl.cfg.

Starting with 15.7 ESD #4, you can also use the SYBOCS_TCL_CFG environment variable to specify the full path to libtcl.cfg.

Setting other environment variables

❖ **Setting environment variables other than SYBASE**

1. Select the appropriate environment variable in the Environment Variables box. The name you select appears in the Variable Name box.
2. Enter the value for the selected environment variable in the Value box.
3. Click Set.

See Appendix A, “Environment Variables,” for more information.

Clearing environment variables

❖ **Clearing environment variables other than SYBASE**

1. Select the appropriate environment variable name in the Environment Variables box. The name you select appears in the Variable Name box.
2. Click Clear.

Configuring a directory driver

Click the Directory Services tab to display the dialog box for configuring directory drivers. The ocscfg utility displays the location of *libtcl.cfg*, the driver configuration file, at the top of the dialog box.

Adding a directory driver entry

❖ **Adding a directory driver entry**

1. Select your platform from the Platform box.
2. Click Add at the bottom of the dialog box. The Add Directory Service Entry dialog box appears.
3. Enter the directory service name in the Directory Service Name box. You can name this element anything as long as it:
   - Contains only letters, numbers, and underscore characters
   - Has a maximum of 64 characters
4. Select a driver from the Directory Service Driver box.
5. Enter a DIT base value in the Directory Service DIT base box. The DIT base is the location where the directory service begins its search for the server entry. For required syntax, see “DIT base syntax” on page 44.
6. Click OK.

DIT base syntax

When adding or modifying a directory driver entry, you can specify a DIT base. DIT base syntax depends on the directory driver that you choose.

Table 7-1: Directory service DIT base syntax

Directory service	DIT base syntax
Windows Registry	These are two examples of Registry DIT base settings: SOFTWARE\SYBASE\SERVER *machine_name*:SOFTWARE\SYBASE\SERVER In the second example, *machine_name* represents a workstation's network name. All DIT base entries must be relative to *HKEY_LOCAL_MACHINE*\. Key entries must exist for the DIT base key and all keys between *HKEY_LOCAL_MACHINE*\ and the DIT base key. The Sybase installation program creates the *HKEY_LOCAL_MACHINE\SOFTWARE\SYBASE* key. For the examples above, you need to add the *SERVER* key. Use the Microsoft regedt32 tool to create any necessary keys. Registry path names are not case sensitive.

If you do not specify a DIT base, the directory driver uses the default value, SOFTWARE\SYBASE\SERVER.

Modifying an existing directory driver entry

❖ **Modifying an existing directory driver entry**

1 Select your platform from the Platform box.

2 Select the appropriate directory service name in the Directory Service Name field.

3 Click Edit at the bottom of the dialog box. The Edit Directory Service Entry dialog box appears.

4 Update the directory service name, driver, and DIT base as required.

5 Click OK.

Deleting a directory driver entry

❖ **Deleting a directory driver entry**

1 Select your platform from the Platform box.

2 Select the appropriate directory service name in the Directory Service Name field.

3 Click Delete.

Activating a directory driver

ocscfg displays the active directory driver in the Active Directory Service box. The first driver listed is the active driver.

❖ **Activating a directory driver**

1 Select the appropriate directory service name in the Directory Service Name field.

2 Click Set Active.

Configuring a security driver

Click the Security Service tab to display the dialog box for configuring security drivers. The ocscfg utility displays the location of *libtcl.cfg*, the driver configuration file, at the top of the dialog box.

Adding a security driver entry

❖ **Adding a security driver entry**

1 Select your platform from the Platform box.

2 Click Add. The Add Security Service Entry dialog box appears.

3 Type the security service name in the Local Name box.

The local name of the security service must correspond to an entry in *objectid.dat*. See “The objectid.dat file” on page 87 for more information.

4 Select a driver from the Security Service Driver box.

5 Click OK.

Modifying an existing security driver entry

❖ **Modifying an existing security driver entry**

1. Select your platform from the Platform box.
2. Select the appropriate security service name in the Local Name field.
3. Click the Edit button at the bottom of the dialog box. The Edit Security Service Entry dialog box appears.
4. Update the security service name and driver as required.
5. Click OK.

Deleting a security driver entry

❖ **Deleting a security driver entry**

1. Select your platform from the Platform box.
2. Select the appropriate security service name in the Local Name field.
3. Click Delete.

Setting the default security driver

The ocscfg utility displays the default security driver in the Default Local Name field. The first driver listed is the default driver.

❖ **Setting the default security driver**

1. Select the appropriate security service name in the Local Name field.
2. Click Set Default.

CHAPTER 8

Using *dsedit*

This chapter explains how to use dsedit to configure a directory service or *sql.ini*.

Using dsedit

The dsedit utility lets you configure a directory service or *sql.ini*.

You can start dsedit from its program icon, the DOS prompt, or the File Manager. You can also start dsedit from the Start menu or Explorer.

- To start dsedit from its program icon, double-click the dsedit icon in the Sybase program group.
- To start dsedit from the DOS prompt, enter:

  ```
  dsedit
  ```

  You can specify the following command line arguments:

Argument	Description
-d dsname	Specifies which directory service to connect to. *dsname* is the local name of the directory service, as listed in the *libtcl.cfg* file. If you do not specify the -d dsname argument, dsedit presents a list of directory service options in the first dialog box.

Argument	Description
-l path	Specifies the path to the *libtcl.cfg* file, if other than *%SYBASE%\%SYBASE_OCS%\ini*. Use this argument only if you want to use a *libtcl.cfg* file other than the one located in *%SYBASE%\%SYBASE_OCS%\ini*.

- To start dsedit through the File Manager or Explorer:
  a Go to the *%SYBASE%\%SYBASE_OCS%\bin* directory.
  b Double-click *dsedit.exe*.
- To start dsedit from the Start menu, choose Start | Programs | Sybase | dsedit.

Opening a session

The Select Directory Service dialog box allows you to open a session with a directory service. You can open a session using one of the following:

- Any directory service that has a driver listed in *libtcl.cfg*
- *sql.ini*

To open a session, do one of the following:

- In the DS Name box, double-click the local name of the directory service to which you want to connect.
- Click the local name of the directory service to which you want to connect and click OK.

Note dsedit uses the SYBASE environment variable to locate *libtcl.cfg*. If you do not set the SYBASE environment variable correctly, dsedit will not locate *libtcl.cfg*.

The session number and local name of the directory service appear in the header bar.

Opening additional sessions

The dsedit utility allows you to have multiple sessions open.

❖ **Opening additional sessions**

1 Choose Open Directory Service from the File menu.

The Select Directory Service box appears.

2 Double-click the local name of the directory service to which you want to be connected, or click the directory service name and click OK.

Opening multiple sessions allows you to copy entries between directory services. See "Copying server entries" on page 56 for more information.

Activating sessions

You must activate a session before you can work in it. To activate a session, do one of the following:

- Click in the session window.
- Choose the session from the Window menu.

The top dsedit header bar shows which session is active.

Adding a server to the directory services

Warning! Most LDAP servers have an ldapadd utility for adding directory entries. Sybase recommends that you use dsedit instead, as it has built-in semantic checks that generic tools do not provide.

dsedit is a graphical utility that allows you to add, delete and modify servers in the *libtcl*.cfg* and *sql.ini* files. Before you can add, delete, or modify an LDAP server entry, you must add the LDAP URL to the *libtcl*.cfg* file. See "The libtcl.cfg and libtcl64.cfg files" on page 68.

❖ **Adding a server to the directory service using dsedit**

Use dsedit to add server to the directory service:

1 From the Microsoft Windows task bar, select Start | Programs | Sybase | dsedit.

2 Select LDAP from the list of servers, and click OK.

3 Click Add New Server Entry.

4 Enter:

- The server name – required.
- Security mechanism – optional. A list of security mechanism OIDs are located in *%SYBASE%\ini\objectid.dat*.
- HA server name – optional. This is the name of the high-availability failover server, if you have one.

5 Click Add New Network Transport.

- Select the transport type from the drop-down list.
- Enter the host name.
- Enter the port number.

6 Click OK twice to exit the dsedit utility.

To view the server entries, enter the following URL in a Web browser:

```
ldap://host:port/ditbase??one
```

For example:

```
ldap://huey:11389/dc=sybase,dc=com??one
```

Note Microsoft Internet Explorer does not recognize LDAP URLs.

Making and modifying server entries

Once you open a session with a directory service or *sql.ini*, you can add, modify, rename, and delete server entries associated with that session.

The server entries associated with the session appear in the Server box. Click a server entry to select it.

Each server entry is made up of a set of attributes. The attributes and attribute values of a server entry, shown in Table 8-1, appear on the right side of the dialog box.

Table 8-1: Server attributes

Attribute name	Type of value	Description	Default value
Server Entry Version	Integer	The version level of the server object definition. Sybase provides this attribute to identify future changes to the object definition.	15501
Server Name	Character string	The server's name.	N/A
Server Service	Character string	A description of the service provided by the server. This value can be any meaningful description.	ASE
Server Status	Integer	The operating status of the server. Valid values are: 1 – Active 2 – Stopped 3 – Failed 4 – Unknown	4
Server Address	Character string	One or more addresses for the server. The format of the address varies by protocol, and some protocols allow more than one format. The options are: • TCP/IP (two formats) 1. *computer_name*,*port_number* 2. *ip-address*,*port_number* • Named Pipe *pipe_name*: "\pipe" is a required prefix to all pipe names. Server pipes can only be local. (Local) *pipe**sql**query* (Remote) *computer_name**pipe**sql**query* • IPX/SPX (three formats) 1. *server_name* 2. *net_number*,*node_number*,*socket_number* 3. *server_name*, *socket_number*	N/A
Server HAfailover (optional)	Character string	The name of the High Availability Failover server, if configured.	N/A

Attribute name	Type of value	Description	Default value
Server Security (optional)	Character string	Object identifier strings (OID) that specify the security mechanisms supported by the server. This attribute is optional. If it is omitted, the Open Server allows clients to connect with any security mechanism for which the Open Server has a corresponding security driver. (See Server Library and security services for process details.) See objectid.dat for more information about object identifier strings.	N/A

Adding a server entry

❖ **Adding a server entry**

1 Choose Add from the Server Object menu. The Input Server Name box appears.

2 Type a server name in the Server Name box.

3 Click OK.

The server entry appears in the Server box. To specify an address for the server, you must modify the entry.

Modifying a server entry

❖ **Modifying a server entry**

1 Click a server entry in the Server box.

2 Click the attribute you want to modify in the Attributes box.

3 Choose Modify Attribute from the Server Object menu. A dialog box appears, showing the current value of the attribute.

4 Enter a new value for the attribute or select a value from the drop-down list. See Table 8-1 on page 53 for a description of each attribute.

5 Click OK.

Renaming a server entry

❖ **Renaming a server entry**

1 Click a server entry in the Server box.

2 Choose Rename from the Server Object menu. The Input Server Name box appears.

3 Enter a new name for the server entry in the Server Name box.

4 Click OK.

Deleting entries

❖ **Deleting a server entry**

1 Click a server entry in the Server box.

2 Choose Delete from the Server Object menu.

Using the ping command

❖ **Verifying your network connection with ping**

1 Click a server entry in the Server box.

2 Select the Ping command from the Server Object menu. The Ping dialog box appears.

3 Click the address you want to ping.

4 Click Ping.

A message box notifies you whether the connection is successful or if the connection fails. If the connection fails, see “Troubleshooting connection failures” on page 59.

Copying server entries

The dsedit utility allows you to copy server entries within a session and between sessions. This includes copying entries from an *sql.ini* file to a directory service.

Copying entries within a session

❖ **Copying server entries within the current session**

1 Click one or more server entries in the Server box. Use the Shift key to select multiple entries.

2 Click the Copy button (below the menu bar), or choose Copy from the Edit menu.

3 Click the Paste button (below the menu bar) or choose Paste from the Edit menu.

dsedit appends the copied server entries with a version number of *_n*. You can rename the copied server entries using the Rename command in the Server Object menu. See "Renaming a server entry" on page 55 for more information.

Copying entries between sessions

❖ **Copying server entries between sessions**

1 Using the directory service or *sql.ini*, open a session to which you want the entries copied.

To open a session, choose Open Directory Service from the File menu. See "Opening additional sessions" on page 50 for more information.

2 Click one or more server entries in the Server box of the session from which you want the entries copied. Use the Shift key to select multiple entries.

3 To copy the server entries, click Copy, or choose Copy from the Edit menu. To cut the server entries, click Cut, or choose Cut from the Edit menu.

4 Activate the session to which you want to paste the server entries. See "Activating sessions" on page 51 for instructions for activating a session.

5 Click Paste, or choose Paste from the Edit menu.

You can rename the copied server entries using the Rename command in the Server Object menu. See “Renaming a server entry” on page 55 for more information.

Exiting *dsedit*

To exit dsedit, choose Exit from the File menu.

CHAPTER 9

Troubleshooting with *dsedit*

This chapter explains how to use dsedit, Sybase's directory services utilities, to test the network connection between a Client-Library application and an Adaptive Server, or Open Server application.

How *dsedit* works

dsedit allows you to verify that your Net-Library software has been installed correctly and that you can connect to an Adaptive Server, or an Open Server application. dsedit mimics the interaction of the Client-Library ct_connect routine and Net-Library but does not require a valid user name on an Adaptive Server or an Open Server application.

You can run dsedit anytime after the Net-Library installation is complete.

To test a server connection, ping the server using dsedit. For instructions, see Chapter 8, "Using dsedit."

Troubleshooting connection failures

If your application fails to connect to a server, run dsedit. Reviewing the messages that dsedit displays may provide you with enough information to solve the problem.

Some types of problems are not diagnosed by dsedit; these are usually problems in your Adaptive Server or Open Server setup, rather than in a Net-Library-to-network-software connection. Troubleshooting suggestions are included in "If dsedit succeeds but other applications fail" on page 61.

If *dsedit* fails

If dsedit does not make a successful connection, make sure that all the basic Net-Library requirements have been met:

- Adaptive Server or Open Server is running on a server machine.
- A network hardware connection exists between your PC and the server machine.
- Your PC meets minimum hardware and software requirements.
- The network vendor's software is installed and running on your PC.
- The connection information in *sql.ini* is correct.

If these requirements have been met, determine the point at which dsedit failed by reviewing the messages it displays.

If dsedit cannot connect to a server, it displays a message box.

If dsedit finds the connection information but notifies you that the server is not responding:

- Verify that the server is running. If you have access to the machine running the server, try using isql to log in to the server. Otherwise, ask your System Administrator to verify that the server you need is running.
- Verify that the networking software and hardware are properly configured. For example, check connectors, plugs, and so on, and confirm that your network software is running.
- Check for any network error messages displayed in the message boxes, or check the system event log for errors.
- Ask your System Administrator to verify that your connection information provides the correct values to connect to the machine running the server, or use the utilities provided with your network software to verify this.

If you cannot resolve the problem yourself, have your designated Sybase support contact call Sybase Technical Support to report the problem. See "Information you need for Sybase Technical Support" on page 61.

If *dsedit* succeeds but other applications fail

If dsedit does not report any errors but your other applications fail to run:

- Verify whether your application uses the default server. If it passes a server name in the ct_connect routine, make sure you selected the corresponding server from the dsedit list before you tested the connection.
- Verify that you have a valid user login name for Adaptive Server or the Open Server application, and that your permissions on databases and tables are consistent with the permissions required to run your applications.
- Use the isql utility to verify that you can access your Adaptive Server or Open Server application. isql is described in *Open Client and Open Server Programmers Supplement*.
- On the machine that is running Adaptive Server or the Open Server application, use isql to verify that the databases and tables used by your application exist. If you do not have access to the machine running Adaptive Server or the Open Server application, or if you are unfamiliar with isql, ask your database administrator to check for you.

Information you need for Sybase Technical Support

If you experience problems with a Net-Library product and must call Sybase Technical Support, be ready to provide the following information:

- The name and version number of your networking software.
- The name and version number of the operating system on which your networking software is running.
- The name and version number of the operating system on which the server to which you are connected is running.
- The version number of the server to which you are connected.
- The date and size of your Net-Library DLL. This information may be obtained by executing the DIR command and displaying a file list that includes the DLL.

Commonly asked questions

- *Question*: I just got a new version of one of the Sybase DLLs. Why does my software still exhibit the old behavior?

  Answer: Verify that only one copy of the DLL exists on your machine. If you find a second DLL with the same name, check your path to see which directory is listed first; you may be inadvertently loading the older version of the DLL.

- *Question*: Why does cs_ctx_alloc fail?

  Answer: Open *sybinit.err* to find a detailed description about why cs_ctx_alloc failed. *sybinit.err* is in the directory in which the application resides.

- Question: Why does ct_init fail?

  Answer: Open the *sybinit.err* file to find a detailed description about why cs_init failed. You can find this file in the directory in which the application resides.

  Verify that all of the drivers listed in *libtcl.cfg* are installed, and that the path to the files is listed in *wsybset.bat*.

- Question: When running isql or dsedit, why do I get a “File Error” stating that it cannot find a DLL that is not one supplied by Sybase?

  Answer: The DLL is probably a network vendor DLL, and the message may mean that your network is not installed properly.

APPENDIX A

Environment Variables

This appendix describes environment variables that contain configuration information.

Environment variables used for connection

Open Client and Open Server use the environment variables shown in Table A-1 during the connection process.

Table A-1: Environment variables used for connection

Variable	Value	Default	Used by
DSLISTEN	The name of the Open Server application, as listed in *sql.ini* or directory service. If DSLISTEN is not set, Open Server uses the default value "SYBASE."	SYBASE	Open Server. Uses DSLISTEN only if the Open Server application does not specify a server during initialization.
DSQUERY	The name of the target server, as listed in *sql.ini* or directory service. If DSQUERY is not set, Open Client uses the default value "SYBASE."	SYBASE	Open Client. Uses DSQUERY only if the Open Client application does not specify the name of the server.
SYBASE	The location of the Sybase installation directory.	Home directory of the "sybase" user	Open Client and Open Server.
SYBASE_OCS	Home directory for the Open Client and Open Server products.	OCS-15_0	Open Client and Open Server.

Variable	Value	Default	Used by
PATH	The directory paths that Open Client and Open Server products search to find executables and DLLs.	Executables	Open Client and Open Server.

Environment variables used for localization

Open Client and Open Server use the environment variables shown in Table A-2 during localization.

Table A-2: Environment variables used for localization

Environment variable	Set it to a locale name that indicates	Used during
LC_ALL	Language, character set, and collating sequence to use for messages, datatype conversions, and sorting.	Initial localization, custom localization
LANG	Language, character set, and collating sequence to use for messages, datatype conversions, and sorting. Open Client and Open Server products search for LANG if they cannot find LC_ALL.	Initial localization
LC_COLLATE	Collating sequence (sort order) to use when sorting and comparing character data.	Custom localization
LC_CTYPE	Character set to use for datatype conversions.	Custom localization
LC_MESSAGE	Language to use for messages.	Custom localization
LC_TIME	Date and time data representation to use for a datetime string, such as date and time formats, names in the native language, and month and day abbreviations.	Custom localization

The LC_* environment variables are POSIX standard environment variables and can be used by non-Sybase applications. Verify that the *locales.dat* file lists the same locale names as are used by the environment variables of the non-Sybase application.

Environment variables used for configuration

Open Client and Open Server products use the environment variables shown in Table A-3 during the configuration process.

Table A-3: Environment variables used for configuration

Environment variable	Description	Used during
SYBOCS_CFG	Overrides the *%SYBASE%\%SYBASE_OCS%\ini\ocs.cfg* default external configuration file path.	Runtime
SYBOCS_DBVERSION	Externally configures the DB-Library version level at runtime. DB-Library uses this variable to retrieve the environment variable at the DB-Library initialization stage and store the environment variable value as the version level. For more information, see the *Open Client DB-Library/C Reference Manual*.	Runtime
SYBOCS_DEBUG_FLAGS	Enables specific diagnostic subsystems. You can enable multiple debug options by specifying a comma-delimited list of flags in the variable. For information about debugging, see the *Open Client Client-Library/C Reference Manual*.	Runtime
SYBOCS_DEBUG_ LOGFILE	Specifies the log file where the diagnostics are recorded. If you do not set this, messages are written to stdout. See the *Open Client Client-Library/C Reference Manual.*	Runtime
SYBOCS_TCL_CFG	Sets the full path name of the *libtcl.cfg* and *libtcl64.cfg* files. For example: `setenv SYBOCS_TCL_CFG c:\joe\libtcl.cfg`	Runtime

APPENDIX B

Configuration Files

This appendix describes the files that Open Client and Open Server products use to obtain configuration information.

About configuration files

Configuration files are created during installation at a default location in the Sybase installation directory structure.

Table B-1 shows the configuration files that Open Client and Open Server products use:

Table B-1: Names and locations for configuration files

Filename	Description	Location	For more information
libtcl.cfg	The driver configuration file contains information regarding directory, security, and network drivers and any required initialization information.	*%SYBASE%\%SYBASE_OCS%\ini* **Note** Use the CS_LIBTCL_CFG property or SYBOCS_TCL_CFG environment variable to specify an alternate path to *libtcl.cfg* file.	See "The libtcl.cfg and libtcl64.cfg files" on page 68. Also see the *Open Client and Open Server Common Libraries Reference Manual*.
sql.ini	This file contains network and security information for each server listed in the file. It is also used as a backup to the *libtcl.cfg* file.	*%SYBASE%\ini*	See "The sql.ini file" on page 74.

Filename	Description	Location	For more information
objectid.dat	This file maps global object identifiers to local names for character set, collating sequence, and security mechanisms.	*%SYBASE%\ini*	See Appendix C, "Localization."
ocs.cfg	The runtime configuration file allows you to change certain Open Client application values at runtime.	*%SYBASE%\%SYBASE_OCS%\ini*	See "The ocs.cfg file" on page 77.

The *libtcl.cfg* and *libtcl64.cfg* files

The *libtcl.cfg* and the *libtcl64.cfg* files (collectively *libtcl*.cfg* files) are the driver configuration files that contain information about the following types of drivers used by Open Client and Open Server products:

- Directory drivers
- Security drivers

A driver is a Sybase library that provides Open Client and Open Server software with a generic interface to an external service provider. This allows Open Client and Open Server to easily support multiple service providers.

Open Client and Open Server reads *libtcl*.cfg* when loading a network, directory, or security driver. An entry in *libtcl.cfg* provides Open Client and Open Server products with the name of the driver and its initialization information.

The purpose of the *libtcl*.cfg* files is to provide configuration information such as driver, directory, and security services for Open Client and Open Server, and for Open Client and Open Server-based applications. Both *libtcl.cfg* and *libtcl64.cfg* are provided on 64-bit platforms. 32-bit applications (on 64-bit platforms) such as dsedit and srvbuild look up the *libtcl.cfg* file while 64-bit applications look up the *libtcl64.cfg* file for configuration information.

The *libtcl*.cfg* file determines whether the *sql.ini* file or LDAP directory services should be used. If LDAP is specified in the *libtcl*.cfg* file, the *sql.ini* file is ignored unless the application specifically overrides the *libtcl*.cfg* file by passing the -I parameter while connecting to a server.

Layout of *libtcl.cfg*

libtcl.cfg is divided into sections, one for each type of driver. ocscfg creates the section headings as follows:

Section heading	Description
NT_DIRECTORY	Lists the Microsoft Windows directory driver
SECURITY	Lists the Microsoft Windows security driver

Note The sections do not have to be in a specific order.

Directory drivers

The syntax for a directory driver entry is:

```
provider=driver ditbase
```

where:

- *provider* is a local name of the directory service. You can name this element anything as long as it contains only letters, numbers, and underscores and has a maximum of 64 characters.
- *driver* is the name of the Microsoft Windows Registry driver called LIBSYBDREG. The default location for drivers is in *%SYBASE%\%SYBASE_OCS%\dll*.
- *ditbase* is where the directory service begins its search for the server entry. The syntax for *ditbase* depends on the directory service provider:

Directory service	DIT base syntax

Microsoft Windows Registry	Following are two examples of Registry DIT base settings:
	`ditbase=SOFTWARE\SYBASE\SERVER`
	`ditbase=`*`machine_name`*`:SOFTWARE\SYBASE\SERVER`
	In the second example, *machine_name* represents a workstation's network name.
	All DIT base entries must be relative to *HKEY_LOCAL_MACHINE*\. Key entries must exist for the DIT base key and all keys between *HKEY_LOCAL_MACHINE*\ and the DIT base key.
	Use the Microsoft regedt32 tool to create any other necessary keys. Registry entries are not case sensitive.

DCE directory service ditbase syntax

If you use DCE directory services, DIT base information in the *libtcl.cfg* file uses this syntax:

ditbase=/.:/dce_cell_name

For example:

```
ditbase=/.:/subsys/sybase/dataservers
```

For LDAP entries in the DIRECTORY section

In its simplest form, LDAP directory services are specified in this format:

```
[DIRECTORY]
ldap=libsybdldap.dll ldapurl
```

where the *ldapurl* is defined as:

```
ldap://host:port/ditbase
```

The following LDAP entry, using these same attributes, is an anonymous connection and only works only if the LDAP server allows read-only access.

```
ldap=libsybdldap.dll
ldap://test:389/dc=sybase,dc=com
```

You can specify a user name and password in the *libtcl*.cfg* file as extensions to the LDAP URL to enable password authentication at connection time.

To set the user name, enter:

```
if (ct_con_props(conn, CS_SET, CS_DS_PRINCIPAL,
ldapprincipal,
     strlen(ldapprincipal), (CS_INT *)NULL) !=
CS_SUCCEED)
{
     ...
}
```

To set the password, enter:

```
if (ct_con_props(conn, CS_SET, CS_DS_PASSWORD,
ldappassword,
     strlen(ldappassword), (CS_INT *)NULL) !=
CS_SUCCEED)
{
     ...
}
```

Encrypting the password

Entries in the *libtcl.cfg* and *libtcl64.cfg* files are in human-readable format. Sybase provides a pwdcrypt utility for basic password encryption. pwdcrypt is a simple algorithm that, when applied to keyboard input, generates an encrypted value that can be substituted for the password. pwdcrypt is located in *%SYBASE%\%SYBASE_OCS%\bin*.

❖ **Encrypting the password**

1 From the Open Client and Open Server (OCS) directory, enter at your command prompt:

```
bin/pwdcrypt
```

2 Enter your password twice when prompted. pwdcrypt generates an encrypted password, for example:

```
0x01312a775ab9d5c71f99f05f7712d2cded2i8d0ae1ce78868d0e8669313d1bc4c706
```

3 Copy and paste the encrypted password into the *libtcl*.cfg* file using any standard ASCII-text editor. Before encryption, the file entry appears as follows:

Note The LDAP URL must be on a single line.

```
ldap=libsybdldap.dll
ldap://dolly/dc=sybase,dc=com????bindname=cn=Manager,dc=sybase,
dc=com?secret
```

4 Replace the password with the encrypted string:

```
ldap=libsybdldap.dll
ldap://dolly/dc=sybase,dc=com????bindname=cn=Manager,dc=sybase,dc=com?
0x01312a775ab9d5c71f99f05f7712d2cded2i8d0ae1ce78868d0e8669313d1bc4c706
```

Warning! Even if your password is encrypted, you should still protect it using file-system security.

Security drivers

Following is the syntax for a security driver entry:

```
provider=driver init_string
```

where:

- *provider* is the local name for the security mechanism. The local name of the security mechanism listed in the object identifiers file, *%SYBASE%\ini\objectid.dat*.

  See "The objectid.dat file" on page 87 for information about the *objectid.dat* file.

- *driver* is the name of the driver. The default location for drivers is in *%SYBASE%\%SYBASE_OCS%\dll*. The possible *driver* values are:

Driver name	Description
libsybsmssp.dll	Microsoft Windows LAN Manager driver
libsybskrb.dll	Kerberos security driver

- *init_string* is an initialization string for the security driver. This element is optional. The value for *init_string* varies by driver.

  For the Kerberos driver, *init_string* specifies the optional qualifier for the security principal names. The syntax for *init_string* is as follows, where *realm* is the value to append to a principal name if the realm information is not available. If the realm name does not start with an "at" sign (@), a forward slash (/) is inserted between the principal name and the realm information.

  ```
  secbase=realm
  ```

Security service initialization syntax

Open Client and Open Server support the Kerberos security driver. To use the Kerberos security driver, perform one of the following:

- Use the ocscfg utility to make an addition to the Security Services.
- Edit the *libtcl.cfg* directly in the *%SYBASE%\%SYBASE_OCS%\ini* directory.

Using the ocscfg utility

To use ocscfg, navigate to the Security Services tab and click Add. Complete the dialog box:

- *Local Name*: Enter *csfkrb5*, or the name you assigned to the Kerberos driver in the *objectid.dat* file.
- *Security Service Driver*: Choose LIBSYBSKRB from the Security Service Init String menu.

When you have entered these two items, click OK. The entry should now appear in the dialog box on the Security Services tab.

Editing libtcl.cfg

If you prefer to edit the *libtcl.cfg* file directly, set the *provider* value for the Kerberos security driver to *csfkrb5*, or to the value you assigned to the Kerberos security driver in the *objectid.dat* file. Set the *driver* value to LIBSYBSKRB. You need to provide an initialization string in the *libtcl.cfg* of the form:

```
secbase=@your_realm_name
```

where *your_realm_name* is the realm where your Kerberos principal is located. This entry is required on Microsoft Windows. For example:

```
[SECURITY]
csfkrb5=LIBSYBSKRB secbase=@SYBASE_CYBER_REALM
```

See Appendix C, “Localization,” for information on the *objectid.dat* localization file.

DCE security service initialization syntax

If you use DCE security service, initialization string information in the *libtcl.cfg* file uses this syntax:

secbase=/.../*dce_cell_name*

For example:

```
secbase=/.../dsatestcell
```

libtcl.cfg example

```
[NT_DIRECTORY]
 ntreg_dsa=LIBDREG  ditbase=software\sybase\serverdsa


[SECURITY]
 NTLM=LIBSMSSP
```

Editing *libtcl.cfg*

Use ocscfg to configure drivers in the *libtcl.cfg* file. See Chapter 7, "Using ocscfg," for instructions for using ocscfg.

The *sql.ini* file

The *sql.ini* file contains information about the network locations of servers. Open Client and Open Server use *sql.ini* as a limited-function directory service. *sql.ini* also serves as a default if an external directory service fails. By default, Open Client and Open Server products look for *sql.ini* in the *%SYBASE%\ini* directory.

- Open Client uses the network information provided by the query line of a *sql.ini* file entry to connect to the server.
- Open Server uses the network information provided by the master line of a *sql.ini* file entry to listen for client connection requests.

An application can specify a different location for Open Client and Open Server products to look for *sql.ini*. See ct_config in the *Open Client Client-Library/C Reference Manual* and srv_props in the *Open Server Server-Library/C Reference Manual* for more information.

Use dsedit to edit *sql.ini*. See Chapter 8, "Using dsedit," for instructions about using dsedit.

sql.ini entries

A *sql.ini* file entry has the form:

```
[SERVERNAME]
service_type=driver,address
secmech=mechanism1,...,mechanismn
```

where:

- *SERVERNAME* is an alias by which Open Client or Open Server recognizes which *sql.ini* entry to read. *SERVERNAME* must begin with a letter (ASCII a-z, A-Z); can contain letters, numbers, and underscores only; and have a maximum of 11 characters.
- *service_type* specifies the type of connection.

  For Microsoft Windows, the options for *service_type* are:

  - "master" for a master line, which is used by server applications to listen for client queries.
  - "query" for a query line, which is used by client applications to find servers.

  The master line and the query line of a *sql.ini* entry contain identical information. dsedit creates both types of lines for each entry. The resulting entry can be used by both clients and servers.
- *driver* is the name of the network driver to use for the connection. A list of network drivers are:

Driver	Description
NLWNSCK	Winsock TCP/IP driver
NLMSNMP	Named Pipes driver
NLNWLINK	SPX/IPX driver

- *address* is the network address for the specified server. The format of the address information depends on the network protocol used for the connection. The options for *address* are:

Protocol	Format(s)	Examples
TCP/IP	Two formats: 1. *computer_name,port_number* 2. *ip-address port_number*	TEST,8877 130.214.30.25,8877
Named Pipe	*pipe_name*: "\pipe" is a required prefix to all pipe names. Server pipes can only be local. (Local) *\pipe\sql\query* (Remote) *\\computer_name\pipe\sql\query*	
IPX/SPX	Three formats: 1. *server_name* 2. *net_number,node_number,socket_number* 3. *server_name, socket_number*	TEST 16,1,83BD TEST,83BD

- "secmech" is the identifier used to list the security mechanisms that a server supports. The "secmech" line is optional.

  See "Security mechanisms" on page 31 for more information about the secmech line.

- *mechanism1,..., mechanism* are the security mechanisms that a server supports. You can specify multiple security mechanisms by using a comma (",") as a separator.

  A security mechanism is listed as its object identifier. An object identifier is a globally unique series of numbers that maps to the local name for a security mechanism in the global object identifiers file.

  See "The objectid.dat file" on page 87 for more information about object identifiers.

sql.ini examples

The following table lists *sql.ini* examples for each protocol:

Protocol	Example

TCP/IP	`[SYBASE]`
	`master=NLWNSCK,TEST,8877`
	`query=NLWNSCK,TEST,8877`

Named Pipe	`[SYBASE]` `master=NLMSNMP,\PIPE\SQL\`QUERY` `query=NLMSNMP,\\TEST\PIPE\SQL\`UERY`

Multiple connection service entries

A server can listen for clients over multiple networks. Clients can connect to servers over multiple networks at runtime.

Servers listening over multiple networks

Edit the server's *sql.ini* file to listen over multiple networks by creating multiple "master" entries, one for each network the server will listen on. For example, the server "MYSERVER" has the following *sql.ini* entry:

```
MYSERVER
    master = NLWNSCK,mercury,1234
    master = NLNWLINK,my_mercury_spx
```

When a server parses *sql.ini* and sees the server name MYSERVER, it listens at the TCP/IP address "mercury,1234" for incoming TCP/IP connections and at the SPX bindery address "my_mercury_spx" for incoming IPX/SPX connections.

Clients connecting over multiple networks

Edit the client's *sql.ini* file to connect over multiple networks by creating multiple "query" entries, one for each network to which the client will connect. For example, the server "SERVER99" has the following "query" services in its *sql.ini* entry:

```
SERVER99
    query = NLWNSCK,mercury,1234
    query = NLWNSCK,plato,9876
    query = NLMSNMP,\\plato\pipe\sql\query
    query = NLNWLINK,my_mercury_spx
```

An Open Client application tries to connect first to the server at "mercury,1234." If that attempt fails, it tries the server at "plato,9876." If that attempt fails, it tries the server at "\\plato\pipe\sql\query" using the Named Pipes protocol. As a final attempt, it tries the server at "my_mercury_spx" using the IPX/SPX protocol. If the final attempt fails, Open Client returns an error.

The *ocs.cfg* file

The *ocs.cfg* runtime configuration file is used by Client-Library applications to set:

- Property values
- Server option values
- Server capabilities
- Debugging options

By using *ocs.cfg*, applications eliminate the need to call routines to set values. A benefit of using *ocs.cfg* is that the application's settings can be changed without recompiling the code.

Client-Library does not read *ocs.cfg* by default. The application must set properties to enable Client-Library to use this file.

See “Using the Open Client and Open Server Runtime Configuration File” in the *Open Client Client-Library/C Reference Manual* for information about the file syntax and the properties you can set in the file.

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

APPENDIX C

Localization

Localization is the process of initializing an application to execute using a specific language and related cultural conventions.

This appendix discusses localization and localization files from a system configuration perspective. For a discussion of programming issues related to localization, see the *Open Client and Open Server International Developers Guide*.

Overview of the localization process

Open Client and Open Server applications can localize in two different ways:

- Using initial localization values
- Using initial localization values and custom localization values

All Open Client and Open Server applications use initial localization values, which are determined at runtime.

In addition, an Open Client and Open Server application can use custom localization values if there is a need to localize at a specific point during the application's execution. Custom localization values override the initial localization values that are set up at runtime.

Environment variables used during localization

Open Client and Open Server use environment variables to determine which locale name to look for in *locales.dat*. When setting up initial localization values, Open Client and Open Server search for the following environment variables:

- LC_ALL
- LANG, if LC_ALL is not set

When setting up custom localization values, Open Client and Open Server may also search for one or more of the following environment variables:

- LC_ALL
- LC_COLLATE
- LC_TYPE
- LC_MESSAGE
- LC_TIME

See the *Open Client and Open Server International Developers Guide* for more information about what environment variables an application uses during custom localization.

See Appendix A, “Environment Variables,” for reference information about the environment variables listed above.

Before running a localized application:

- Make sure the *locales.dat* file contains an entry which reflects the localization values the application will use. If it does not, add an appropriate entry.
- Make sure that the localization files that your application will use are installed:
  - Localized message files are located in the *%SYBASE%\locales\message* directory.
  - Collating sequence files are located in the *%SYBASE%\charsets* directory.

  Open Client and Open Server products come with the localization files to support one language, and one or more character sets and sort orders.

Localization files

At runtime, Open Client and Open Server applications pick up localization information from external files. Three directories in the Sybase release directory contain these files:

- The *locales* directory contains:
  - The *locales.dat* file, which maps locale names to languages, character sets, and collating sequences.
  - The *message* subdirectory, which contains localized error messages for Open Client and Open Server, organized by language name.
  - *language_name* subdirectories, which are included to provide compatibility with previous releases of Open Client and Open Server software. These directories contain localized message files organized by character set.
  - The *unicode* directory, which contains error message files for system management utilities.
- The *charsets* directory contains a subdirectory for each supported character set. Each subdirectory contains sort and conversion files for the character set.
- The *ini* directory contains:
  - The *objectid.dat* file, which maps a global identifier for an object to local platform-specific names.

All Open Client and Open Server products include files to support at least one language and one or more character sets and collating sequences. During installation, these files are loaded into the Sybase home directory structure in the correct locations.

When configuring an Open Client or Open Server application, you must verify that the above directories contain the correct files for your site and application.

The *locales* directory

The *locales* directory contains files that your application uses to load localization information. It also contains language-specific message files.

The locales.dat file

Located in the *%SYBASE%\locales* directory, *locales.dat* provides platform-specific locale information in a Sybase proprietary format. This file associates locale names with languages, character sets, and collating sequences.

Warning! If you plan to use isql, which uses iso_1 as its client character set default when talking to a server, modify the character sets to prevent data corruption by performing *one* of the following:

- Add a new entry to the section of the *locales.dat* file, such as isql.german.cp850, and call isql with option -J isql.
- Set LANG=isql, to change the client character set to cp850.
- Issue a command like mode con cp SELECT=1250 before calling isql, so that the display character set is changed to iso_1.

How *locales.dat* is used

Open Client and Open Server applications use *locales.dat* to determine what localization information to load. *locales.dat* directs Open Client and Open Server applications to localization information, but it does not contain actual localized messages or character set information.

locales.dat sections and entries

locales.dat contains platform-specific sections, each of which contains predefined locale definition entries. These entries vary by platform, but all sections include an entry defining a “default” locale.

Locale definition entries have the form:

```
locale = locale_name, language_name, charset_name
[,sortorder_name]
```

where:

- *locale_name* is the name of the locale definition. The default values for *locale_name* are vendor-specified and based on POSIX terminology. Comments at the end of the *locales.dat* file list POSIX values for locale names.
- *language_name* is the subdirectory name by which Sybase products recognize the language.

- *charset_name* is the subdirectory name by which Sybase products recognize the character set.
- *sortorder_name* is the file name by which Sybase products recognize the collating sequence (optional).

The following *locales.dat* file entry specifies a French locale. Because no sort order is specified, the default sort order "binary" is used with this locale:

```
locale = fr.FR.88591, french, iso_1
```

locales.dat example

The following portion of a *locales.dat* file illustrates a platform-specific section in a *locales.dat* file:

```
[NT]
locale = enu, us_english, cp1252
locale = fra, french, cp1252
locale = deu, german, cp1252
locale = default, us_english, cp1252
```

Editing *locales.dat*

If the predefined entries in *locales.dat* do not meet your needs, use a text editor to edit the file.

Warning! Before you edit, make a copy of the original *locales.dat*. The copy will help you solve any problems with the edited version. Also, review the entries for your platform to see if an entry already exists.

You can:

- Change the "default" locale definition.
- Add a locale definition.
- Match a locale name used by non-Sybase software. For example, the Sybase predefined locale name is "fr":

  ```
  locale = fr, french, iso_1
  ```

  If a non-Sybase application requires a value of "french" for the LC_ALL environment variable, change the locale name to:

  ```
  locale = french, french, iso_1
  ```

To add a new entry to *locales.dat* or to change an existing entry:

1 Choose a value for *locale_name*.

2 Determine the value for *language_name*.

When a Sybase language module is installed, a subdirectory for the language is created in the *locales\message* directory of the Sybase directory tree. *language_name* must correspond to this subdirectory's name.

3 Determine the value for *charset_name*.

When a Sybase language module is installed, subdirectories for each supported character set are created in the *charsets* directory of the Sybase directory tree. *charset_name* must correspond to one of these subdirectory names.

4 Determine the value for *sortorder_name* if you want a sort order other than binary.

The *charsets\charset_name* subdirectory contains the sort order (*.*srt*) files for the character set. *sortorder_name* must correspond to one of these file names (without the *.srt*).

5 In the appropriate platform-specific section of the *locales.dat* file, enter or change the appropriate entry.

After you make the change:

- Update localization environment variables (LC_ALL, LC_CTYPE, LC_MESSAGE, LC_TIME, LANG) as appropriate.
- If you have added a new locale name and you want existing applications to use this new name in cs_locale calls, edit and recompile the applications as appropriate.

You need not delete entries from *locales.dat*, even if applications no longer use them. If you decide to delete an entry, make sure no application uses it.

Localized message files

Warning! Do not edit localized message files.

Localized message files contain product messages in a particular language. These message files (the **.loc* files in the *%SYBASE%\locales\message\language_name* directories) enable Open Client and Open Server applications to generate messages in a variety of languages.

All Open Client and Open Server products include English (us_english) message files. Your products may also include files to support additional languages.

If you purchase and install a new language module, the installation process adds a *language_name* subdirectory containing message files in the new language.

Message file names sometimes vary by platform, but most resemble the following names:

- *cslib.loc* – CS-Library messages
- *ctlib.loc* – Client-Library messages
- *oslib.loc* – Server-Library messages
- *blklib.loc* – Bulk Library messages
- *bcp.loc* – Bulk Copy messages
- *esql.loc* – Embedded SQL messages

All Open Client and Open Server message files use the Unicode ISO 10646 UTF-8 character set.

Open Client and Open Server products convert messages from UTF-8 to other character sets as needed.

The *charsets* directory

The *charsets* directory contains collating sequence files for each supported character set and a *unicode* directory, which contains conversion files used by Unilib®.

Collating sequence files

Warning! Do not edit collating files.

The order in which a system sorts characters is called its collating sequence or sort order.

Open Client and Open Server products include files to support a variety of collating sequences. These files, located in the *%SYBASE%\charsets* directory, can vary by platform but generally include the following:

- *binary.srt*
- *dictionary.srt*
- *noaccents.srt*
- *nocase.srt*
- *nocasepref.srt*

Collating sequences are specified in *locales.dat* file entries. If a *locales.dat* file entry does not specify a collating sequence, then a binary sort order is used.

For more information about collating sequences, see the *Open Client and Open Server International Developers Guide*.

Unicode conversion files

Unicode conversion files contain conversion configuration information in Unicode character set (ISO 10646) in UTF-8 form. These conversion files are available for each Sybase-supported character set.

The *ini* directory

The *ini* directory contains the global object identifiers file (*objectid.dat*).

The objectid.dat file

The global object identifiers file, called *objectid.dat*, associates a unique global object identifier with the local name of an object.

An object identifier is a series of non-negative integer values separated by a dot. An object identifier is based on a naming tree defined by the international standards bodies CCITT and ISO.

Location of *objectid.dat*

objectid.dat is located in the *Sybase_home\locales* directory.

objectid.dat sections and entries

objectid.dat contains a section for each class of object.

Object class entries have the form:

```
[Object Class]
    object_identifier local_name1, ..., local_namen
```

where:

- *Object Class* is the section identifier.
- *object_identifier* is the globally unique object identifier.
- *local_name1,..., local_namen* are the local names associated with the object identifier, separated by a comma.

objectid.dat example

The following portion of an *objectid.dat* file illustrates sections in *objectid.dat*:

```
[charset]
     1.3.6.1.4.1.897.4.9.1.1 = iso_1
     1.3.6.1.4.1.897.4.9.1.2 = cp850
     1.3.6.1.4.1.897.4.9.1.3 = cp437
     1.3.6.1.4.1.897.4.9.1.4 = roman8
     1.3.6.1.4.1.897.4.9.1.5 = mac

[collate]
     1.3.6.1.4.1.897.4.9.3.50 = binary
     1.3.6.1.4.1.897.4.9.3.51 = dictionary
     1.3.6.1.4.1.897.4.9.3.52 = nocase
     1.3.6.1.4.1.897.4.9.3.53 = nocasepref
     1.3.6.1.4.1.897.4.9.3.54 = noaccents
```

```
[secmech]
     1.3.6.1.4.1.897.4.6.1 = dce, dcesecmech
     1.3.6.1.4.1.897.4.6.3 = NTLM, N, ntsecmech
     1.3.6.1.4.1.897.4.6.6 = csfkrb5, kerberos
```

Note If you change the local name of an object, use a text editor to edit *objectid.dat* accordingly.

APPENDIX D

Secure Sockets Layer in Open Client and Open Server

This appendix describes the SSL support for Open Client and Open Server and summarizes some system configuration tasks that are required to use the SSL protocol.

For an overview of the Open Client and Open Server security services architecture, see Chapter 6, “Using Security Services.”

SSL handshake

SSL is an industry standard for sending wire- or socket-level encrypted data over client-to-server and server-to-server connections. Before the SSL connection is established, the server and the client exchange a series of I/O round trips to negotiate and agree upon a secure, encrypted session. This is called the “SSL handshake.”

When a client application requests a connection, the SSL-enabled server presents its certificate to prove its identity before data is transmitted. Essentially, the SSL handshake consists of the following steps:

- The client sends a connection request to the server. The request includes the SSL (or Transport Layer Security, TLS) options that the client supports.
- The server returns its certificate and a list of supported CipherSuites, which includes SSL/TLS support options, the algorithms used for key exchange and digital signatures.

- A secure, encrypted session is established when both client and server have agreed upon a CipherSuite.

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

For a list of CipherSuites that Open Client and Open Server supports, see the *Open Client Client-Library/C Reference Manual*.

SSL security levels and security mechanisms

Security levels

SSL provides several levels of security in Open Client and Open Server:

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

SSL filter as a security mechanism

When establishing a connection to an SSL-enabled Adaptive Server, the SSL security mechanism is specified as a filter on the master and query lines in the *sql.ini* file. SSL is used as an Open Client and Open Server protocol layer that sits on top of the TCP/IP connection.

The SSL filter is different from other security mechanisms, such as DCE and Kerberos, which are defined with SECMECH (security mechanism) lines in the *sql.ini* file. The master and query lines determine the security protocols that are enforced for the connection.

A typical *sql.ini* file on Microsoft Windows using SSL looks like this:

```
[SERVER]
    master=TCP,hostname,address1, ssl
    query=TCP,hostname,address1, ssl
```

where *hostname* is the name of the server to which the client is connecting, and *address1* is the port number of the host machine. All connection attempts to a master or query entry in the *sql.ini* file with an SSL filter must support the SSL protocol. A server can be configured to accept SSL connections and have other connections that accept plain text (unencrypted data) or use other security mechanisms.

For example, an Adaptive Server *sql.ini* file that supports both SSL-based connections and plain-text connections looks like this:

```
[SYBSRV1]
    master=NLWNSCK,hostname,2748,ssl
    query=NLWNSCK,hostname,2748,ssl
    master=NLWNSCK,hostname,2749
    query=NLWNSCK,hostname,2749
```

In this example, the SSL security service is specified on port number 2748. On SYBSRV1, Adaptive Server listens for clear text on port number 2749, which is without any security mechanism or security filter.

Validating a server by its certificate

Any Open Client and Open Server connection to an SSL-enabled server requires that the server have a certificate file, which consists of the server's certificate and an encrypted private key. The certificate must also be digitally signed by a CA.

Open Client applications establish a socket connection to Adaptive Server similarly to the way that existing client connections are established. Before any user data is transmitted, an SSL handshake occurs on the socket when the network transport-level connect call completes on the client side and the accept call completes on the server side.

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

- The SSL-enabled server must present its certificate when the client application makes a connection request.
- The client application must recognize the CA that signed the certificate. A list of all "trusted" CAs is in the trusted roots file. See "The trusted roots file" on page 93.
- For connections to SSL-enabled servers, the common name in the server's certificate must match the server name in the *sql.ini* file as well.

When establishing a connection to an SSL-enabled Adaptive Server, Adaptive Server loads its own encoded certificates file at start-up from *%SYBASE%\%SYBASE_ASE%\certificates\servername.crt*. The *servername* is the name of the Adaptive Server as specified on the command line when starting the server with the -S flag or from the server's environment variable DSLISTEN.

Other types of servers may store their certificate in a different location. See the vendor-supplied documentation for the location of your server's certificate.

Common name validation in an SDC environment

The default behavior for SSL validation in Open Client and Open Server is to compare the common name in the server's certificate with the server name specified by ct_connect(). In a shared disk cluster (SDC) environment, a client may specify the SSL certificate common name independent of the server name or the SDC instance name. A client may connect to an SDC by its cluster name—which represents multiple server instances—or to a specific server instance.

Open Client and Open Server supports common name validation in an SDC environment. This allows the Adaptive Server SSL certificate common name to be different from the server or cluster name by allowing the client to use the transport address to specify the common name used in the certificate validation. The transport address can be specified in one of the directory services like the *interfaces* file, LDAP or NT registry, or through the connection property CS_SERVERADDR.

This is the syntax of the server entries for the SSL-enabled Adaptive Server and cluster for Microsoft Windows:

```
[CLUSTERSSL]
query=tcp,hostname1,5000, ssl="CN=name1"
query=tcp,hostname2,5000, ssl="CN=name2"
query=tcp,hostname3,5000, ssl="CN=name3"
query=tcp,hostname4,5000, ssl="CN=name4"

[ASESSL1]
master=tcp,hostname1,5000, ssl="CN=name1"
query=tcp,hostname1,5000, ssl="CN=name1"

[ASESSL2]
master=tcp,hostname2,5000, ssl="CN=name2"
query=tcp,hostname2,5000, ssl="CN=name2"

[ASESSL3]
master=tcp,hostname3,5000, ssl="CN=name3"
query=tcp,hostname3,5000, ssl"CN=name3"

[ASESSL4]
master=tcp,hostname4,5000, ssl="CN=name4"
```

```
query=tcp,hostname4,5000, ssl="CN=name4"
```

The trusted roots file

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

The trusted roots file for Open Client and Open Server is located in *%SYBASE%\ini\trusted.txt*.

Currently, the recognized CAs are Thawte, Entrust, Baltimore, VeriSign, and RSA.

By default, Adaptive Server stores its own trusted roots file in *%SYBASE%\%SYBASE_ASE%\certificates\servername.txt*.

Both Open Client and Open Server allow you to specify an alternate location for the trusted roots file:

- Open Client:

  ```
  ct_con_props (connection, CS_SET, CS_PROP_SSL_CA,
  "SYBASEhome\ini\trusted.txt", CS_NULLTERM, NULL);
  ```

  where *SYBASEhome* is the installation directory. CS_PROP_SSL_CA can be set at the context level using ct_config, or at the connection level using ct_con_props.

- Open Server:

  ```
  srv_props (context, CS_SET, SRV_S_CERT_AUTH,
  "SYBASEhome\ini\trusted.txt", CS_NULLTERM, NULL);
  ```

  where *SYBASEhome* is the installation directory.

bcp and isql utilities also allow you to specify an alternative location for the trusted roots file.The parameter -x is included in the syntax, allowing you to specify an alternative location for the *trusted.txt* file.

Obtaining a certificate

The System Security Officer installs signed server certificates and private keys in the server. You can get a server certificate by:

- Using third-party tools provided with existing public-key infrastructure already deployed in the customer environment
- Using the Sybase certificate request tool in conjunction with a trusted third-party CA

To obtain a certificate, you must request a certificate from a CA. If you request a certificate from a third-party and that certificate is in PKCS #12 format, use the certpk12 utility to convert the certificate into a format that is understood by Open Client and Open Server. See “certpk12” on page 102.

To test the certificate request tool and to verify that the authentication methods are working on your server, Open Client and Open Server provide the certreq and certauth tools that allow you to function as a CA and issue a CA-signed certificate to yourself.

The main steps to creating a certificate for use with a server are:

1 Generate the certificate request.

2 Generate the public and private key pair.

3 Securely store the private key.

4 Send the certificate request to the CA.

5 After the CA signs and returns the certificate, append the private key to the certificate.

6 Store the certificate in the server’s installation directory.

Using third-party tools to obtain a certificate

Most third-party PKI vendors and some browsers have utilities to generate certificates and private keys. These utilities are typically graphical wizards that prompt you through a series of questions to define a distinguished name and a common name for the certificate.

Follow the instructions provided by the wizard to create certificate requests. Once you receive the signed PKCS #12-format certificate, use certpk12 to generate a certificate file and a private key file. Concatenate the two files into a *servername.crt* file, where *servername* is the name of the server, and place it in the server's installation directory. By default, the certificates for Adaptive Server's are stored in *%SYBASE%\%SYBASE_ASE%\certificates.* See "certpk12" on page 102.

Using Sybase tools to request and authorize certificates

Sybase provides tools for requesting and authorizing certificates that are available in the *%SYBASE%\%SYBASE_OCS%\bin* directory. certreq generates public and private key pairs and certificate requests. certauth converts a server certificate request to a CA-signed certificate.

Warning! Use certauth only for testing purposes. Sybase recommends that you use the services of a commercial CA because it provides protection for the integrity of the root certificate, and because a certificate that is signed by a widely accepted CA facilitates the migration to the use of client certificates for authentication.

Preparing a server's trusted root certificate is a 5-step process. Perform all 5 steps to create a test trusted root certificate so you can verify that you are able to create server certificates. When you have a test CA certificate (trusted roots certificate), repeat steps 3 through 5 to sign server certificates.

❖ **Preparing a server's trusted root certificate**

1 Use certreq to request a certificate.

2 Use certauth to convert the certificate request to a CA self-signed certificate (trusted root certificate).

3 Use certreq to request a server certificate and private key.

4 Use certauth to convert the certificate request to a CA-signed server certificate.

5 Append the private key text to the server certificate and store the certificate in the server's installation directory.

Note certauth and certreq are dependent on RSA and DSA algorithms. These tools only work with vendor-supplied crypto modules that use RSA and DSA algorithms to construct the certificate request.

The following reference sections describe the tools used in the previous steps.

For information on adding, deleting, or viewing server certificates on Adaptive Server, see the *Adaptive Server Enterprise System Administration Guide*.

certauth

Converts a server certificate request to a CA- (certificate authority) signed certificate.

Syntax

certauth
[-r]
[-C *caCert_file*]
[-Q *request_filename*]
[-K *caKey_filename*]
[-N *serial_number*]
[-O *SignedCert_filename*]
[-P *caPassword*]
[-s *start_time*]
[-T *valid_time*]
[-v]

Parameters

-r

when specified, creates a self-signed root certificate for the test environment.

-C *caCert_file*

specifies the name of the CA's certificate request file when -r is specified, or specifies the name of the CA's root certificate.

-Q *request_filename*

specifies the name of certificate request file.

-K *caKey_filename*

specifies the name of the CA's private key.

-N *serial_number*

specifies the serial number in the signed certificate. If -N is not specified, certauth generates a pseudo-random serial number.

-O *SignedCert_filename*

specifies the name to use for the output when creating a signed certificate file. If -r is specified, *SignedCert_filename* is the self-signed root certificate. If -r option is not used, *SignedCert_filename* is the certificate signed by the *caCert_file*.

-P *caPassword*

specifies the CA's password that is used to decrypt its private key.

-s *start_time*

specifies the start of the validity period for the certificate, from the current time, in units of days. The default start time is the current time, unless specified with -s.

-T *valid_time*

specifies the length of the validity period for the certificate, in units of days.

-v

prints the version number and copyright message of the certauth tool, then exits.

Example 1

This example converts the CA's certificate request (*ca_req.txt)* to a certificate, using the private key (*ca_pkey.txt*). The private key is protected using *password*. This example sets the valid time range to 365 days, self-signs the certificate, and outputs it as a root certificate (*trusted.txt*).

```
certauth -r -C ca_req.txt -Q ca_req.txt
-K ca_pkey.txt -P password -T 365 -O trusted.txt
```

The utility returns this message:

```
 -- Sybase Test Certificate Authority --
Certificate Validity:
    startDate = Tue Sep 5  10:34:43  2000

    endDate = Wed Sep 5  10:34:43  2001
```

```
CA sign certificate SUCCEED (0)
```

Note You need to create a trusted root certificate for the test CA only once. After you have created the trusted root certificate, you will use it to sign many server certificates in your test environment.

Example 2

This example converts a server certificate request (*srv5_req.txt*) to a certificate, and sets the valid time range to 180 days. This example signs the certificate with a CA's certificate and private key (*trusted.txt* and *ca_pkey.txt*), uses password protection, and outputs the signed certificate as *sybase_srv5.crt*.

```
certauth -C trusted.txt -Q srv5_req.txt
-K ca_pkey.txt -P password -T 180 -O sybase_srv5.crt
```

Note If you do not set a valid time, the default is 365 days.

The utility returns this message:

```
 -- Sybase Test Certificate Authority --
Certificate Validity:
    startDate = Tue Sep  5 10:38:32  2000

    endDate = Sun Mar  4 09:38:32  2001

CA sign certificate SUCCEED (0)
```

Below is a sample certificate. See the Usage section below for additional steps to take to create a server certificate that the server can use.

```
-----BEGIN CERTIFICATE-----

MIICSTCCAgUCAVAwCwYHKoZIzjgEAwUAMG8xCzAJBgNVBAYTAlVTMRMwEQYDVQQI
EwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpFbWVyeXZpbGxlMQ8wDQYDVQQKFAZTeWh
c2UxDDAKBgNVBAsUA0RTVDEXMBUGA1UEAxQOc3liYXNlX3Rlc3RfY2EwHhcNMDAw
ODE4MTkxMzM0WhcNMDEwODE4MTkxMzM0WjBvMQswCQYDVQQGEwJVUzETMBEGAUE
CBMKQ2FsaWZvcm5pYTETMBEGA1UEBxMKRW1lcnl2aWxsZTEPMA0GA1UEChQGU3li
YXNlMQwwCgYDVQQLFANEU1QxFzAVBgNVBAMUDnN5YmFzZV90ZXN0X2NhMIHwMIo
BgcqhkjOOAQBMIGcAkEA+6xG7XCxiklxbP96nHBnQrTLTCjHlcy8QhIekwv9OlqG
EMG9AjJLxj6VCkPOD75vqVMEkaPPjoIbXEJEe/aYXQIVAPyvY1+B9phC2e2YFcf7
cReCcSNxAkBHt7rnOJZ1Dnd8iLQGt0wd1w4lo/Xx2OeZS4CJW0KVKkGId1hNGz8r
GrQTspWcwTh2rNGbXxlNXhAV5g4OCgrYA0MAAkA70uNEl90Kmhdt3RISiceCMgOf
1J8dgtWF15mcHeS8OmF9s/vqPAR5NkaVk7LJK6kk7QvXUBY+8LMOugpJf/TYMAsG
AhUAhM2Icn1pSavQtXFzXJUCoOmNLpkCFQDtE8RUGuo8ZdxnQtPu9uJDmoBiUQ==

-----END CERTIFICATE-----
```

Usage

- The maximum length of the serial number in the -N option is 20 hexadecimal characters. If the specified serial number is longer, certauth truncates the serial number to the maximum length.
- To create a server certificate file that Adaptive Server understands, append the certificate requestor's private key to the end of the signed certificate file. Using the example above, you would cut and paste *srv5_pkey.txt* to the end of the signed certificate file, *sybase_srv5.crt*.
- To create a trusted roots file that the server can load at start-up, rename *trusted.txt* to *sybase_srv5.txt*, where *sybase_srv5.txt* is the common name of the server.
- Then copy the *sybase_srv5.txt* file into the Adaptive Server installation directory, for example, *%SYBASE%\%SYBASE_ASE%\certificates*.

The file, which is required for an SSL-based session, is used to start the SSL-enabled Adaptive Server.

After the CA's root certificate is created, it can be used to sign multiple server certificates.

See also

certreq

certreq

Creates a server certificate request and corresponding private key. This utility can be used in interactive mode, or you can provide all optional parameters on the command line.

Syntax

certreq
[-F *input_file*]
[-R *request_filename*]
[-K *PK_filename*]
[-P *password*]
[-v]

Parameters

-F *input_file*

specifies the input-file name that contains attribute information to build a certificate request. If you do not specify an *input_file* name, the required information must be interactively entered by a user.

The *input_file* needs an entry for each of the following:

```
req_certtype={Server,Client}
req_keytype={RSA,DSA}
```

```
req_keylength={for RSA: 512-2048;
     for DSA: 512,768,1024}
req_country={string}
req_state={string}
req_locality={string}
req_organization={string}
req_orgunit={string}
req_commonname={string}
```

Note The common name must be the same as the server name, except in a cluster environment where multiple servers may use the same common name.

See "Common name validation in an SDC environment" on page 92 for more details.

See *Example 2* for a sample file, called *input_file*.

-R *request_filename*

specifies the name for the certificate-request file.

-K *PK_filename*

specifies the name for the private-key file.

-P *password*

specifies the password used to protect the private key.

-v

displays the version number and copyright message, then exits.

Example 1

This example does not use the -F *input_file* parameter, therefore, it is in interactive mode. To create a server certificate request (*server_req.txt*) and its private key (*server_pkey.txt*), enter:

```
certreq

Choose certificate request type:
     S - Server certificate request
     C - Client certificate request (not supported)
     Q - Quit
Enter your request [Q] : s

Choose key type:

     R - RSA key pair
     D - DSA/DHE key pair
     Q - Quit
```

```
Enter your request [Q] : r

Enter key length (512, 768, 1024 for DSA; 512-2048 for
RSA) : 512

Country: US

State: california

Locality: emeryville

Organization: sybase

Organizational Unit: dst

Common Name: server
```

The utility returns the message:

```
Generating key pair (please wait) . . .
```

After the key pair is generated, the certreq utility prompts you for more information.

```
Enter password for private key : password

Enter file path to save request: server_req.txt

Enter file path to save private key : server_pkey.txt
```

Example 2

Alternatively, you can use the -F option for noninteractive mode. When you use the -F option, use valid values and follow the format described above. Failure to do so prevents the certificate from being built correctly.

Below is a sample text file that can be used for noninteractive entry for a certificate request.

```
certreq -F input_file

req_certtype=server
req_keytype=RSA
req_keylength=512
req_country=us
req_state=california
req_locality=emeryville
req_organization=sybase
req_orgunit=dst
req_commonname=server
```

After you create and save this file, enter on the command line:

```
certreq -F path_and_file -R server_req.txt
-K server_pkey.txt -P password
```

where *path_and_file* is the location of the text file.

This file creates a server certificate request, *server_req.txt*, and its private key, *server_pkey.txt*, which is protected by *password*.

You can edit the server certificate file with any standard ASCII text editor.

Usage

- The input file uses the format of <tag>=*value*. <tag> is case sensitive and should be the same as described above.
- The "=" is required. Valid *value* should start with a letter or digit, must be a single word, and there should not be any spaces within *value*.
- *value* is required for <tag>s "req_certtype," "req_keytype," "req_keylength," and "req_commonname."
- The space or tab around <tag>, "=" and *value* is allowed. Blank lines are also allowed.
- Each comment line should start with "#".
- The certificate request file is in PKCS #10 format and used as acceptable input for the certauth tool to convert the request to a CA-signed certificate.

See also

certauth

certpk12

Exports or imports a PKCS #12 file into a certificates file and a private key.

Syntax

certpk12
{-*O Pkcs12_file* | -*I Pkcs12_file*}
[-C *Cert_file*]
[-K *Key_file*]
[-P *key_password*]
[-E *Pkcs12_password*]
[-v]

Parameters

-C *Cert_file*

specifies the name of certificate file to be exported to a PKCS #12 file if -O is on; or the name of certificate file to be imported from a PKCS #12 file if -I is on.

-K *Key_file*

specifies the name of private key file to be exported to a PKCS #12 file if -O is on; or the name of private key file to be imported from a PKCS #12 file if -I is on.

-P *Key_password*

specifies the password which is used to protect the private key specified by -K. If -O is on, the password is required to export the private key to a PKCS #12 file; if -I is on, the password is required to output the private key to a text file after it is imported from a PKCS #12 file.

-O *Pkcs12_file*

specifies the name of a PKCS #12 file to be exported. The file can contain a certificate plus a private key, a single certificate, or a single private key. Either -O or -I needs to be on.

-I *Pkcs12_file*

specifies the name of a PKCS #12 file to be imported. The file can contain a certificate plus a private key, a single certificate, or a single private key. Either -I or -O needs to be on.

-E *Pkcs12_password*

specifies the password used to protect the PKCS #12 file. If -O is on, the password is used to encrypt the PKCS #12 file to be exported; if -I is on, the password is used to decrypt the PKCS #12 file to be imported. The password is also called "transport password."

-v

prints the version number and copyright message of the certpk12 tool and exits.

Example 1

This example exports certificate file, *caRSA.crt* and private key file, *caRSApkey.txt* to a PKCS #12 file, *caRSA.p12*. *password* is the password used to decrypt *caRSApkey.txt*. *pk12password* is the password used to encrypt the final *caRSA.p12*:

```
certpk12 -O caRSA.p12 -C caRSA.crt -K caRSApkey.txt
     -P password -E pk12password

-- Sybase PKCS #12 Conversion Utility certpk12 Thu Nov
9 16:55:51 2009--
```

Example 2

This example imports a PKCS #12 file, *caRSA.p12* which contains a certificate and a private key. Output the embedded certificate to a text file, "*caRSA_new.crt*" and the embedded private key to a text file, "*caRSApkey_new.txt*". *new_password* is used to protect *caRSApkey_new.txt*, and *pk12password* is required to decrypt *caRSA.p12* file:

```
certpk12 -I caRSA.p12 -C caRSA_new.crt
  -K caRSApkey_new.txt  -P new_password
-E pk12password

-- Sybase PKCS#12 Conversion Utility certpk12 Thu Nov 9
16:55:51 2009--
```

Note After running examples 1 and 2, *caRSA.crt* and *caRSA_new.crt* are identical. *caRSApkey.txt* and *caRSApkey_new.txt* are different because they are encrypted randomly.

Example 3

This example exports the certificate file, *caRSA.crt* to a PKCS#12 file, *caRSAcert.p12*. *pkcs12password* is used to encrypt *caRSAcert.p12*:

```
certpk12 -O caRSAcert.p12 -C caRSA.crt
-E pk12password

-- Sybase PKCS#12 Conversion Utility certpk12 Thu Nov 9
16:55:51 2009--
```

Example 4

This example imports a PKCS #12 file, *caRSAcert.p12* which contains a certificate. Output the embedded certificate to a text file, *caRSAcert.txt*. *pk12password* is required to decrypt *caRSAcert.p12* file.

```
certpk12 -I caRSAcert.p12 -C caRSAcert.txt
    -E pk12password

-- Sybase PKCS#12 Conversion Utility certpk12 Thu Nov 9
16:55:51 2009--
```

Note After running examples 3 and 4, *caRSA.crt* and *caRSAcert.txt* are identical.

Usage

- certpk12 only supports triple-DES encrypted PKCS #12 file.
- Append certificate requestor's private key to the end of its signed certificate file.
- Name the file *servername*.crt, where *servername* is the name of the server, and place it in the certificates directory under *%SYBASE%\%SYBASE_ASE%*.

  This file is needed to start the SSL-enabled Adaptive Server.

See also

certreq and certauth

FIPS 140-2 compliance for password encryption

Encryption of login and remote passwords in Open Client and Open Server is accomplished with the Sybase Common Security Infrastructure (CSI). Certicom SSL Plus 5.2.2 CSI-Crypto 2.6 complies with the Federal Information Processing Standard (FIPS) 140-2. To support FIPS encryption, a Certicom Security Builder shared library named *sbgse2.dll* is installed in *%SYBASE%/%SYBASE_OCS%/lib3p* or in *%SYBASE%/%SYBASE_OCS%/lib3p64* when you install SDK or Open Server.

Index

E

F

G

H

I

L

N

O

P

S

T

U

V