SYBASE®
An SAP® Company

Configuration Guide

Open Client™ and Open Server™

15.7

[UNIX]

DOCUMENT ID: DC35831-01-1570-02

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 UNIX* contains information about configuring your system to run Open Client™ and Open Server™. For a list of operating system platforms the Open Client and Open Server products are available on, see the *New Features Bulletin Open Server and SDK for Windows, Linux, and UNIX*.

Audience

This book is written for System Administrators, Sybase® Database Administrators, and developers. It discusses configuration tasks and topics in terms of system administration rather than application programming.

How to use this book

The *Open Client and Open Server Configuration Guide for UNIX* is divided into three parts:

- Configuration instructions
- Configuration utilities
- Configuration references

Configuration instructions

- Chapter 1, “Configuration Overview,” provides an overview of the configuration process and the configuration requirements.
- Chapter 2, “Basic Configuration for Open Client,” explains how a client application connects to a server and lists the configuration tasks required.
- 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 the steps necessary to configure your Open Client applications to connect to the secondary server during failover.
- Chapter 5, “Using a Directory Service,” explains how applications get connection information from a directory service and lists the 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.

Configuration utilities

- Chapter 7, “Using dscp,” explains how to use the dscp command-line utilities to configure the server entries in directory services and the *interfaces* file.
- Chapter 8, “Using dsedit,” explains how to use the dsedit utility to configure the server entries in directory services and the *interfaces* file. dsedit is an X-Windows-based utility with a graphical user interface.

Configuration references

The configuration topics are divided into appendices by the source of configuration information.

- Appendix A, “Environment Variables,” lists and explains how to set the environment variables that Open Client and Open Server products use.
- Appendix B, “Configuration Files,” presents an overview of configuration files and describes:
  - *libtcl.cfg*, the driver configuration file
  - *interfaces*, the *interfaces* 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, “Kerberos Security Services,” lists the security services supported by the CyberSafe Kerberos security driver and summarizes CyberSafe configuration requirements for use as an Open Client and Open Server security mechanism.
- Appendix E, “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 in order 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 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 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

Before you read this document, install Open Client according to the instructions in the *SDK and Open Server Installation Guide for UNIX*. Open Client is packaged with and is part of the Software Developer's Kit (SDK) or Open Server software.

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

About Open Client and Open Server

Open Client provides three application programming interfaces (APIs), named *dblib* and *ctlib*, and Net-Library.These products enable communications between Adaptive Server Enterprise and Open Server applications and 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 (except for DB-Library), allow communications with clients and other servers. In addition, Open Server provides routines to:

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

Overview of configuration

Open Client and Open Server software requires 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 (except for DB-Library) or Open Server application

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

- Establish a connection with Adaptive Server or an Open Server application

Note Except where noted, information in this document 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 about DB-Library, see the *Open Client DB-Library/C Reference Manual*.

The initialization process

To initialize an application, Open Client and Open Server:

- Use the SYBASE environment variable to determine the location of the Sybase installation directory.

- 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.
- 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.

To establish a connection, Open Client:

- Uses DSQUERY environment variable to determine the name of the target server. Use DSQUERY only if the Open Client application does not specify the name of the target server. If specified in both, the DSQUERY and the application, the application designation takes precedence.
- *Uses a directory service or the interfaces* file to obtain the target server's address.

Note DB-Library can only look up servers using the *interfaces* file.

To listen for a connection request, Open Server:

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

- Uses a directory service or the *interfaces* file to determine 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, including:

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

There are additional tasks if you are using one of the following:

- A directory service
- Security services
- Custom localization values in addition to or instead of initial localization values

The following chapters provide configuration instructions. Refer to the configuration chapter appropriate to your installation.

CHAPTER 2

Basic Configuration for Open Client

This chapter describes 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, it 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 configuration for Open Client

All Open Client applications require some basic configuration information, obtained during initialization and connection, including:

1 (Not applicable to DB-Library) The location of the Sybase installation directory as defined by the SYBASE environment variable.

2 (Not applicable to DB-Library) A locale name. Open Client uses the values of the following POSIX environment variables as locale names:

- 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 neither environment variable is defined, Open Client uses “default” as the locale name.

3 (Not applicable to DB-Library) Localized message and character set files. Open Client looks in the *locales.dat* file for an entry whose name matches the locale name determined previously. Then, it loads the localized messages and character set files specified in the *locales.dat* file.

4 The name of the target server. Open Client obtains the name of the target server from one of the following sources, in this order:

   a The client application, which can provide the server name in the call to ct_connect (or dbopen). Some applications, such as isql, can specify the name of the target server through command line options.

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

   c The default name SYBASE, if DSQUERY is not set.

5 The target server’s network address. Open Client obtains the target server’s addresses from the directory service or from *interfaces*. DB-Library does not examine the *libtcl.cfg* file, it accesses the *interfaces* file:

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

   - *interfaces* – If a directory service is not used, or if it is used and fails, Open Client searches for the SERVERNAME entry in *interfaces* that matches the name as determined previously and uses the corresponding target address.

6 (Not applicable to DB-Library) The name of the security service driver. Open Client looks in the [SECURITY] section of *libtcl.cfg* to determine which security driver to load.

Refer to Chapter 6, “Using Security Services,” for more information on security services.

Note Items 1-3 occur when the Open Client Client-Library application calls the cs_ctx_alloc or cs_ctx_global routine. Items 4 through 6 occur when the Open Client application calls ct_connect.

Configuration tasks for Open Client

To enable Open Client to successfully initialize your client application and carry out connection requests, complete these tasks:

1 Set 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.

See Appendix A, “Environment Variables,” for instructions about how to set environment variables.

2 Set localization files:

Verify that 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 locale name you want.

See Appendix C, “Localization,” for information on localization.

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.”

4 If you want to change the directory driver or security driver, configure *libtcl.cfg:*

- Specify a directory driver in the [DIRECTORY] section of *libtcl.cfg*.
- Specify a security driver in the [SECURITY] section of *libtcl.cfg*.

See Appendix B, “Configuration Files,” for reference information about *libtcl.cfg*.

5 Configure *interfaces* or directory service:

Create a server entry in *interfaces* or LDAP directory service using dscp.

See Chapter 7, “Using dscp,” for instructions about using dscp.

See Chapter 5, “Using a Directory Service,” for information about directory services.

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 what type of application it is. See the *Open Server Server-Library/C Reference Manual* for more information about the types of Open Server applications.

Overview of configuration for Open Server

All Open Server applications require some basic configuration information, obtained during initialization and connection, including:

1 The location of the Sybase installation directory as defined by the SYBASE environment variable.

2 A 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 *locales.dat*. If neither environment variable is defined, Open Server uses "default" as the locale name.

3 Localized message and character set files. Open Server looks in *locales.dat* for an entry whose name matches the locale name determined above. Open Server then loads the localized messages and character set files specified in *locales.dat*.

4 The 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:

- The Open Server application, which can provide the server name in the call to srv_init
- The DSLISTEN environment variable, if the application does not specify its name
- The default name SYBASE, if DSLISTEN is not set

5 The target server's network address. Open Server gets the target server's address(es) from either the directory service or from *interfaces*:

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

interfaces – If a directory service is not used, or if it is used and fails, Open Server searches for the SERVERNAME entry in *interfaces* that matches the name as determined above and uses the corresponding target address.

6 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 enable Open Server to successfully initialize your server application and respond to connection requests, complete these tasks:

1 *Configure libtcl.cfg:*

- Specify a directory driver in the [DIRECTORY] section of *libtcl.cfg*.
- Specify a security driver in the [SECURITY] section of *libtcl.cfg*.

See Appendix B, "Configuration Files," for reference information about *libtcl.cfg*.

2 *Configure interfaces or directory service:*

Create a server entry in *interfaces* or LDAP directory service using dscp.

See Chapter 7, "Using dscp," for instructions about using dscp. See "The interfaces file" on page 68 for reference information about *interfaces*. See Chapter 5, "Using a Directory Service," for information about directory services.

3 *Set 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.

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

- 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 Appendix A, “Environment Variables,” for instructions about how to set environment variables. See Appendix C, “Localization,” for information on localization.

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*. 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 High Availability (HA) Failover. Embedded SQL™/C and Embedded SQL/COBOL support HA Failover starting with version 12.5.1.

Add hafailover line to *interfaces* 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 *interfaces* file to provide the information necessary for the client to connect to the secondary companion. You can add this line using a file editor or the dsedit utility.

The following *interfaces* file entry is for an asymmetric configuration between the primary companion "PERSONNEL1" and its secondary companion "MONEY1." It includes an hafailover entry that enables clients connected to" PERSONNEL1" to reconnect to "MONEY1" during failover:

```
PERSONNEL1
   master tcp ether huey 5000
   query tcp ether huey 5000
   hafailover MONEY1
```

Note Client applications must resend any queries that were interrupted by failover. Other information specific to the connection, such as cursor declarations, will 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 Failover software.

❖ **Modifying an application with Client-Library calls**

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 *interfaces* file has a valid entry for the secondary companion server, the connection becomes a failover connection, and the client reconnects appropriately.

However, if the CS_FAILOVER property is set but the *interfaces* 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. 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 HA 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 *interfaces* file entry.
- Use the -Q command-line option.

If your *interfaces* file contained the example entry given in "Add hafailover line to interfaces file," you can use isql with failover by entering the following:

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

CHAPTER 5

Using a Directory Service

Client-Library and Server-Library applications use directory services to keep track of information about servers. This chapter describes how directory services work and the configuration tasks necessary to use them.

Note DB-Library does not support directory services.

Overview of directory services

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

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

UNIX platforms can use Lightweight Directory Access Protocol (LDAP) directory services.

LDAP directory services

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 *interfaces* 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 *interfaces* file

LDAP directory services are a convenient alternative to the typical Sybase *interfaces* file. The Sybase *interfaces* file stores server information in a "flat" file. Any changes to server information in the *interfaces* file need to be updated on each machine (client and server) in the enterprise.

Table 5-1 highlights the differences between the Sybase *interfaces* file and LDAP server.

Table 5-1: The interfaces file versus LDAP directory services

The *interfaces* 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 *interfaces* file on a UNIX machine with a TCP connection and a failover machine looks like this:

```
master tcp ether huey 5000
query tcp ether huey 5000
hafailover secondary
```

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

```
dn: sybaseServername=foobar, dc=sybase,dc=com
objectClass: sybaseServer
sybaseVersion: 1500
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.
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, NAMEPIPE. • The value of the sybaseStatus. • Address: any valid address for the protocol type. **Note** The dscp utility splits this attribute into Transport type and Transport address.
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/config* directory for the following LDAP services:

- *sybase.schema* - it contains the directory schema to be used with OpenLDAP servers.

- *sybase-schema.conf* - it contains the same 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 between Client-Library and the server begins. If the login attempt fails.Client-Library does not retry any other addresses.

The most important entity is the address attribute, which contains the information for setting up a connection to the server and for 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.

Because 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.

Server objects and attributes

The directory service must contain information about servers accessed by your Open Client. Use dscp to modify *interfaces* and to add servers to an LDAP service.

A directory service identifies a server entry as a *directory object*. Each directory object has a unique set of *attributes*, recognized by Client-Library and Server-Library, as shown in Table 5-3:

Table 5-3: Server attributes

Attribute	Description
Server Object Version	Symbolic integer code for the version of the object definition. Sybase provides this attribute to identify future changes to the object definition.
Server Name	A string value that specifies the server's name. The name can be any string that is 512 or less bytes long. Do not confuse a server name attribute with the name used to locate the directory entry. The latter is the fully qualified name for the directory entry, expressed in the name syntax of the directory provider. To avoid confusion, administrators should verify that the name attribute at least partially matches the server's fully-qualified name (for example, the attribute value could be the entry's common name).
Server Service	A string value that describes the service that the server provides. The service value can be any string that is 512 bytes long or less.
Server Status	Symbolic integer code that describes the operating status of the server. Valid values are: 1 - Active 2 - Stopped 3 - Failed 4 - Unknown
Transport Address	One or more transport addresses for the server. The transport address attribute has two elements: • Transport type • Transport address
Security Mechanism	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" on page 34 for process details.) See "The objectid.dat file" on page 80 for more information about object identifier strings. See the [SECMECH] section of *$SYBASE/config/objectid.dat* for examples.

How applications use a directory service

Client-Library and Server-Library can use a directory service, rather than *interfaces*, to obtain a server's address.

To retrieve information from a directory service, Open Client and Open Server software use a directory driver, a 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.

Client-Library and Server-Library determine whether to use a directory service or *interfaces* as follows:

1 If the 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 60 for information about directory drivers and *libtcl*.cfg*.

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

3 Client-Library and Server-Library fall back and use *interfaces* 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.

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 ct_con_props(). You can override these values by explicitly setting the ct_con_props() for that specific connection.
- The CS_LIBTCL_CFG property specifies the name and path to an alternate *libtcl.cfg* file.

- 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 63.
- 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- and 64-bit applications.

The *libtcl*.cfg* file is located in the *$SYBASE/$SYBASE_OCS/config* directory.

The connection process has these basic steps:

1 Client-Library uses the Sybase directory driver specified in the *libtcl*.cfg* file to request the address of my_server.

2 The directory service looks up the attributes for the my_server entry and returns the information to Client-Library using the Sybase directory driver.

3 The application uses the address to connect to the machine where my_server resides.

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/lib*.

The LDAP SDK library locations and environment variable are listed in Table 5-5 on page 28.

Warning! Sybase LDAP directory services do not support client applications built with DB-Library.

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 60) or set with property calls to Client-Library. The user name and password that are passed to the LDAP server through 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, called a *bind*, can be unsecured or it can have user name and password authentication. The type of access allowed is determined by the server.

Anonymous connections

A connection where 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-only access.

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.

❖ **Using a directory service**

1 Configure the LDAP server according to the vendor-supplied documentation.

2 Add the path environment variable to the LDAP library for your platform, for example:

```
setenv LD_LIBRARY_PATH \
$LD_LIBRARY_PATH:$SYBASE/$SYBASE_OCS/lib3p
```

Note See Table 5-5 on page 28 for a list of environment variables and libraries for your platform.

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 on page 20 for supported LDAP URL values.

Note The LDAP URL must be on a single line.

Following is the context for this entry:

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

For example:

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

where "*one*" indicates the scope of a search that retrieves entries one level below the DIT base.

Note For 64-bit support, replace *lib3p* with *lib3p64* and *libsybdldap.so* with *libsybdldap64.so* for the above examples.

For information on the supported platforms, see Chapter 2, "OpenLDAP," in the *Open Client Client-Library/C Reference Manual*.

Table 5-4 defines the keywords for the *ldapurl* variables.

Table 5-4: 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 appropriate environment variable points to the required third-party libraries. Table 5-5 lists the location of the Netscape LDAP SDK libraries.

Table 5-5: Environment variables

Platform	Environment variable	Library location
HP HP-UX Itanium 32-bit	SHLIB_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
HP HP-UX Itanium 64-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p64*
HP HP-UX PA-RISC 32-bit	SHLIB_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
HP HP-UX PA-RISC 64-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p64*
Linux x86 32-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
Linux x86-64 64-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p64*
Linux POWER 32-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
Linux POWER 64-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p64*
Linux AMD64 (Opteron)/EM64T	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p64*
IBM AIX POWER 32-bit	LIBPATH	*$SYBASE/$SYBASE_OCS/lib3p*
IBM AIX POWER 64-bit	LIBPATH	*$SYBASE/$SYBASE_OCS/lib3p64*
Solaris x86-64 32-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
Solaris x86-64 64-bit	LD_LIBRARY_PATH_64	*$SYBASE/$SYBASE_OCS/lib3p64*
Solaris SPARC 32-bit	LD_LIBRARY_PATH	*$SYBASE/$SYBASE_OCS/lib3p*
Solaris SPARC 64-bit	LD_LIBRARY_PATH_64	*$SYBASE/$SYBASE_OCS/lib3p64*

5 Add your server entry to the LDAP server using dscp or dsedit. See “Adding and modifying server entries” on page 40, 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:

```
[DIRECTORY]
   ldap=libsybdldap.so ldap://test:389/dc=sybase,dc=com
   ldap=libsybdldap.so ldap://huey:11389/dc=sybase,dc=com
```

In this example, if the connection to *test:389* fails, the connection fails over to the LDAP server on *huey:11389*. Different vendors use different DIT-base formats. For more information, 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.so
```

```
ldap://myADhost: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 the 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 set up a secure connection between a client and an LDAP Directory Server, use either of these methods:

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

  ```
  [DIRECTORY]
  ldap=libsybdldap.so
  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 in the *libtcl.cfg* file:

  ```
  [DIRECTORY]
  ldap=libsybdldap.so 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 use the security services provided by third-party security software to authenticate users and protect data transmitted between machines on a network. This chapter describes 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. 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. UNIX platforms can use the security mechanism provided by Kerberos security.

You specify the security mechanisms that a server supports in *interfaces* or a directory service. The values for *interfaces* or directory service's secmech line/attribute must correspond to the strings associated with object identifiers defined in the user's *objectid.dat* file, under the [secmech] section:

- The optional secmech line in an *interfaces* 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 Kerberos driver requires a server using a Kerberos 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* used to establish a secure connection between a client and a server. Each security service addresses a particular security concern.

Security services consist of:

- Authentication services, and
- 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.

See Appendix D, "Kerberos Security Services," for a list of security services provided by Kerberos.

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.

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

1 Client-Library validates the login with the security mechanism. The security mechanism returns a login *token*, which Client-Library sends to the server, along with information about what 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 server application allows the login.

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

1 Client-Library uses the security mechanism to prepare the data packet it will send to the Open Server application. Depending on which security services are requested, the security mechanism encrypts the data or creates 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.

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

Client-Library and security services

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

1 If the client application specifies a security mechanism, Client-Library checks in the [SECURITY] section of *libtcl.cfg* for a matching driver and loads that driver.

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

3 Client-Library determines which security services will be used for the connection from the client application.

If there is no *libtcl.cfg* or there are no entries in the [SECURITY] section, then there is no network security provider. In that case, the Open Server application 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 the [SECURITY] section of *libtcl.cfg*. Administrators can further restrict the list of supported mechanisms by adding a secmech attribute to the directory entry for the server.

When an Open Client application requests a security session from an Open Server application, the following occurs:

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 *interfaces* 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 Appendix B, “Configuration Files,” 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 the [SECURITY] section of *libtcl.cfg*.

Configuration tasks

To enable your Open Client and Open Server application to use security services, you must:

- Configure Kerberos
- Configure libtcl.cfg

The following sections describe each of these tasks.

Configure Kerberos

See Appendix D, “Kerberos Security Services,” and your Kerberos documentation.

Configure *libtcl.cfg*

Specify a security driver in the [SECURITY] section of *libtcl.cfg*.

Note Open Client and Open Server software use the first entry in the [SECURITY] section as the default security driver.

See Appendix B, “Configuration Files,” for reference information about security drivers and *libtcl.cfg*.

Optionally, to restrict the security mechanisms that a server supports:

- If your application uses *interfaces*, add a secmech line in the server’s *interfaces* entry.

- If your application uses a directory service, use the dscp utility to add the secmech attribute to the server's directory service.

See Chapter 7, "Using dscp," for information about adding information to a directory service or an *interfaces* file.

CHAPTER 7

Using dscp

This chapter explains how to use dscp to configure the *interfaces* file and to configure a directory service.

About dscp

The dscp command-line utility allows you to view and edit server entries in the *interfaces* file or a LDAP directory service. After opening a session, you can check your configuration, view existing entries, create new entries, and modify entries as needed. Use this utility if your system does not have X-Windows.

Note The dsedit utility is an X-Windows-based graphical tool that lets you view and edit server entries in the *interfaces* file. For more information, see Chapter 8, “Using dsedit.”

Starting dscp

If you plan to add or modify entries, you must log in to the directory service, with the necessary privileges, before you start dscp.

To start dscp, enter:

```
$SYBASE/$SYBASE_OCS/bin/dscp
```

The dscp prompt, >>, appears. Table 7-1 shows the commands you can use:

Table 7-1: dscp commands

Command	Description
open [*DSNAME*]	Opens a session with the specified directory service or *interfaces*. dscp – to open a session with *interfaces*, specify "InterfacesDriver" as *DSNAME*.
sess	Lists all open sessions.
[switch] SESS	Makes session number *SESS* the current session.
close [SESS]	Closes a session identified by the *SESS* number. If you do not specify *SESS*, closes the current session.
list [all]	Lists the server entries for the current session. To list the names of the entries, use the list command. To list the attributes for each entry, use the list all command.
read *SERVERNAME*	Prints the contents of server entry *SERVERNAME* to the screen.
add *SERVERNAME*	Adds server entry *SERVERNAME* in the current session. dscp prompts you for information about *SERVERNAME*. Press Return to accept the default value, which is shown in brackets [ ].
adtr *SERVERNAME*	Adds an attribute to the server entry *SERVERNAME* in the current session.
mod *SERVERNAME*	Modifies server entry *SERVERNAME* in the current session. dscp prompts you for information about *SERVERNAME*. Press Return to accept the default value, which is shown in brackets [ ].
del *SERVERNAME*	Deletes server entry *SERVERNAME* in the current session.
delete-all	Deletes all server entries in the current session.
copy *NAME1* to {*NAME2* | *SESS* | *SESS NAME2*}	Copies server entry *NAME1* in the current session to: • Server entry *NAME2* in the current session • Session *SESS* • Server entry *NAME2* in session *SESS*
copyall to *SESS*	Copies all server entries in the current session to session *SESS*.
config	Prints configuration information related to your Sybase environment to the screen.
exit, quit	Exits dscp.

Command	Description
help, ?, h	Displays the help screen.

Viewing your configuration

View the current Open Client and Open Server configuration and directory service provider names using the config command.

When you enter:

```
config
```

the dscp utility prints the following information to the screen:

- The value of the SYBASE environment variable
- The location of the driver configuration file
- The names of directory service providers with which you can open a dscp session

Getting help

To view the dscp help screen, enter one of these commands:

```
help
h
?
```

Using dscp sessions

Before you can view, add, or modify server entries, you must open a session: Opening a dscp session allows you to interact with *interfaces*.

You can have multiple sessions open at any one time.

Opening a session

To open a session with *interfaces*, enter:

```
open InterfacesDriver
```

When you open a session, dscp tells you the session's number. For example, if you open a session with *interfaces* using the open InterfacesDriver command, dscp returns the following message:

```
ok
Session 1 InterfacesDriver>>
```

Listing sessions

To list all open sessions, enter:

```
sess
```

Switching between open sessions

To switch to another open session, enter:

```
switch SESS
```

where SESS is the session number. For example, if you enter:

```
switch 3
```

you are switched to session 3. The switch keyword is optional.
If you enter the following:

```
3
```

you are also switched you to session 3.

Closing a session

To close a session, enter:

```
close SESS
```

where *SESS* is the session number. For example, if you enter:

```
close 3
```

session 3 is closed. Use the sess command to list all open sessions.

If you do not specify *SESS*, the current session is closed.

Adding and modifying server entries

After you open a session with a directory service or *interfaces*, you can list, add, modify, and delete associated server entries.

Note When you add or modify a server entry, dscp automatically creates or modifies both master and query lines. The master line and the query line of an *interfaces* file entry contain identical information.

Each server entry is made up of a set of attributes. When you add or modify a server entry, dscp prompts you for information about each attribute. Table 7-2 describes each attribute:

Table 7-2: Server attributes

Attribute	Type of value	Default value	Modifiable when adding or modifying server entry?
Server Entry Version	Integer	15001	*Adding* Directory Services: No *interfaces*: No *Modifying* Directory Services: Yes *interfaces*: No
Server Name	Character string	N/A	*Adding* Directory Services: N/A *interfaces*: N/A *Modifying* Directory Services: No *interfaces*: No
Service	Character string	ASE	*Adding* Directory Services: Yes *interfaces*: Yes *Modifying* Directory Services: Yes *interfaces*: No
Server Status	Integer	4 Valid values are: 1 - Active 2 - Stopped 3 - Failed 4 - Unknown	*Adding* Directory Services: No *interfaces*: No *Modifying* Directory Services: Yes *interfaces*: No
Transport Address • Transport type • Transport address	Transport type: Character string Transport address: Character string	Transport type: tcp Transport address: None. Valid transport type value is “tcp”. Valid transport address: Character string in a format recognized by the specified transport type.	*Adding or modifying* Directory Services: Transport type: Yes Transport address: Yes *interfaces*: Transport type: Yes Transport address: Yes

Attribute	Type of value	Default value	Modifiable when adding or modifying server entry?
Security Mechanism	Character string. *Note:* You can add up to 20 security mechanism strings for each server entry.	None. Valid values are Character strings associated with object identifiers defined in the user's *objectid.dat*.	*Adding* Directory Services: Yes *interfaces*: Yes *Modifying* Directory Services: Yes *interfaces*: Yes
HA Failoverserver (optional)	Character string.	None.	*Adding* Directory Services: Yes *interfaces*: Yes *Modifying* Directory Services: Yes *interfaces*: Yes

Listing server entries

To list the names of server entries associated with a session, enter:

```
list
```

To list the attributes of server entries associated with a session, enter:

```
list all
```

See Table 7-2 for a description of server attributes.

Viewing a server entry

To view the contents of a server entry, enter:

```
read SERVERNAME
```

For example, if you enter:

```
read myserver
```

the following information is displayed:

```
DIT base for object:  interfaces
 Distinguish name: myserver
 Server Version: 1
 Server Name: myserver
 Server Service: ASE
 Server Status: 4 (Unknown)
```

```
Server Address:
    Transport Type: tcp
    Transport Addr: victory 1824
    Transport Type: tcp
    Transport Addr: victory 1828
```

See Table 7-2 for a description of the server attributes listed above.

Adding a server entry

To add a server entry, enter:

```
add SERVERNAME
```

The dscp utility prompts you for information about *SERVERNAME*. Enter a value for each attribute or press return to accept the default value, which is shown in brackets [ ].

For example, if you enter:

```
add myserver
```

The dscp utility prompts you for the following information:

```
Service: [ASE]
 Transport Type: [tcp] tcp
 Transport Address: victory 8001
 Security Mechanism []:
```

To exit the add mode, press Enter until you return to the dscp prompt >>.

A server entry can have up to 20 transport type/address combinations associated with it.

See Table 7-2 for a description of the server attributes listed above.

❖ **Adding a server entry to an LDAP directory service**

To use dscp to make an entry into an LDAP server, you need to enable LDAP by editing the *$SYBASE/$SYBASE_OCS/config/libtcl.cfg* file, and adding the entry of the LDAP server you will be using.

Warning! If you have a blank space after your LDAP server entry, dscp will default to using the interfaces driver and not connect to an LDAP server.

Use dscp to add a server to the directory services.

1 To start dscp, enter:

```
$SYBASE/$SYBASE_OCS/bin/dscp
```

2 Before you can view, add, or modify server entries, you must open a session.

Opening a dscp session allows you to interact with any directory service that has a driver listed in *libtcl*.cfg*. To open a session, enter:

```
open DSNAME
```

where *DSNAME* is the name of the directory service.

If you do not specify *DSNAME*, dscp uses the default directory service provider specified in the *libtcl*.cfg* file. If there are no entries in the *libtcl*.cfg* file, dscp uses the default *interfaces* file located in *$SYBASE*.

3 A connection to an LDAP server is indicated by:

```
Session 1 ldap>>
```

If the LDAP server requires user authentication for login, you must use the -U*username* command-line parameter flag when connecting to the server.

If the LDAP server is configured to allow anonymous access, the user name and password are not required. If the user name and password are specified in the *libtcl*.cfg* file, dsedit and dscp utilities use these variables.

4 To add a server to the directory services, enter:

```
add server_name
```

where *server_name* is the name of the server to be added.

5 The next prompt specifies the service type. Adaptive Server is the default value:

```
Service [ASE Server]
```

Press Enter to accept the default.

6 Enter the transport type. Press Enter to accept the default value of TCP, or enter a value from Table 5-3.

7 Enter the transport address. Valid entries are any values that enable the transport type specified. For example, for a TCP connection, enter: `host_name port_number`.

8 Because any LDAP server entity can have multiple address entries, you are again prompted for “Transport type.” Enter another transport type, or press Enter (leaving the field blank) to skip this prompt and proceed.

9 At the prompt, enter another valid transport address that corresponds to the additional transport type, or press Enter (leaving the field blank) to proceed.

10 Optionally, enter a security mechanism OID.

11 Optionally, enter a secondary server for failover.

12 Press Enter. When completed, the following message displays:

```
Added server_name done
```

To view the server entries, enter the following URL in a Netscape or Mozilla-based 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.

Modifying a server entry

To modify an existing server entry, enter:

```
mod SERVERNAME
```

dscp prompts you for information about *SERVERNAME*. Enter a value for each attribute, or press Return to accept the existing value, which is shown in brackets [ ].

For example, if you enter:

```
mod myserver
```

the dscp utility prompts you for information as follows:

```
Version: [1]
 Service: [ASE] Open Server
 Status: [4]
 Address:
     Transport Type: [tcp]
     Transport Address: [victory 1824] victory 1826
     Transport Type: [tcp]
     Transport Address: [victory 1828]
     Transport Type: []
```

```
Security Mechanism []:
```

Note dscp cannot modify the Version, Service, and Status entries.

To delete an address, enter:

```
>>del SERVERNAME
```

To exit the modify mode, press Enter until you return to the dscp prompt `>>`.

Deleting server entries

You can delete one entry or all entries associated with a session. To delete one entry, enter:

```
del SERVERNAME
```

For example, if you enter:

```
del myserver
```

the dscp utility deletes the entry for "myserver." To delete all entries associated with a session, enter:

```
delete-all
```

Copying server entries

dscp allows you to copy server entries within a session and between sessions. This includes copying entries from *interfaces* to a directory service.

There are four options when copying a server entry. You can:

- Copy a server entry to a new name in the current session
- Copy a server entry to a different session
- Copy a server entry to a new name in a different session
- Copy all entries in the current session to a different session

Copying entries within a session

You can copy a server entry within a session, if you want to create a new server entry. To copy an entry within a session, enter:

```
copy NAME1 to NAME2
```

For example, if you enter:

```
copy myserver to my_server
```

dscp creates a new entry “my_server” identical to “myserver.” You can then modify the new entry and leave the original intact.

Copying entries between sessions

There are two options for copying a server entry between sessions. You can:

- Keep the name of the server entry
- Rename the server entry

To copy an entry to a different session and keep the server name, enter:

```
copy NAME1 to SESS
```

where:

- *NAME1* is the current server name.
- *SESS* is the number of the session to which you want to copy the server entry.

For example, if you enter:

```
copy myserver to 2
```

dscp copies the “myserver” entry in the current session to session 2.

To copy an entry to a different session and give it a different name, enter:

```
copy NAME1 to SESS NAME2
```

where:

- *NAME1* is the current server name.
- *SESS* is the number of the session to which you want to copy the server entry.
- *NAME2* is the new server name.

For example, if you enter:

```
copy myserver to 2 my_server
```

dscp copies the “myserver” entry in the current session to session 2 and renames it “my_server.”

Copying all entries to a different session

To copy all entries in the current session to a different session, enter:

```
copyall to SESS
```

where *SESS* is the number of the session to which you want to copy all entries:

For example, if you enter:

```
copyall to 2
```

dscp copies all entries in the current session to session 2.

Exiting dscp

To exit dscp, enter one of these commands:

```
exit
quit
```

CHAPTER 8

Using dsedit

This chapter explains how to use dsedit to configure *interfaces* and how to configure Sybase server listings in a directory service.

About dsedit

The dsedit X-Windows-based graphical tool lets you view and edit server entries in the *interfaces* file.

If your system does not have X-Windows, use dscp or a simple text editor to configure server entries in *interfaces*. See Chapter 7, “Using dscp,” for more information.

Starting dsedit

If you plan to add or modify servers, make sure that you will be able to edit *interfaces* directory before starting dsedit. To edit *interfaces* entries, you must have write permission on *interfaces*.

To start dsedit, enter:

```
$SYBASE/$SYBASE_OCS/bin/dsedit
```

If you are running dsedit from a remote machine, make sure that the DISPLAY environment variable is set correctly. See your X11 documentation for information on setting the DISPLAY environment variable.

Note To get help on any screen, click Help.

Opening a session

After starting dsedit, you will see the main screen. This screen allows you to select and open editing sessions for *interfaces* files.

Interfaces file sessions

To open the default *interfaces* for editing, select Sybase *interfaces* file and click OK. To open an alternate file, edit the displayed file name before clicking OK. You can open multiple *interfaces* file sessions with different files.

The session window for an *interfaces*-file session displays the full path name of *interfaces* and lists the server entries contained in *interfaces*. The buttons to the right of the list allow you to add, modify, copy, and delete entries:

- Add new server entry – displays the Server Entry Editor window, where you specify the name and network addresses for a new server entry. See “Adding, viewing, and editing server entries” on page 52 for more information.
- Modify server entry – lets you view and modify the network addresses for a selected server entry. Select the server in the list, then click Modify server entry to display the server’s attributes in the Server Entry Editor window. See “Adding, viewing, and editing server entries” on page 52 for more information.
- Copy server entry – lets you copy one or more entries to another *interfaces* file. Before copying server entries, select the entries to copy in the list of servers:
  - To copy a single entry, click it once.

- To copy a range of consecutive entries, click the first (or last) entry in the range, then, with the Shift key pressed, click the last (or first) entry in the range.
- To select multiple, nonconsecutive entries, press and hold down the Ctrl key, then click each desired entry to select it.

After selecting entries to copy, click Copy server entries. A new window prompts you to choose the destination directory service. You can copy to another *interfaces* file:

- To copy the entries to another *interfaces* file, select Sybase Interfaces File, edit the displayed file name, and click OK.

Click Close Session to close the session window and write any changes to *interfaces*.

Note You must close the *interfaces* session window to apply your edits to the *interfaces* file.

Adding a server to the directory services

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

Using dsedit, you can add, delete, and modify servers in the directory service and *interfaces* files. However, 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 60.

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

1 From the *$SYBASE/$SYBASE_OCS/bin* directory, enter:

```
dsedit
```

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

3 Click Add New Server Entry.

4 Enter:

- The server name – this is required.

- Security mechanism – optional. A list of security mechanism OIDs are located in *$SYBASE/config/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 two times to exit the dsedit utility.

To view the server entries, enter the following URL in a supported Web browser or LDAP administration tool:

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

For example:

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

Note Microsoft Internet Explorer does not recognize LDAP URLs.

Adding, viewing, and editing server entries

Use the Server Entry Editor window to view or edit server entries in an *interfaces* file. The Add New Server Entry and Modify Server Entry buttons in the Session window display the Server Entry Editor window and its fields:

- Server name – to add a server entry, type the name of the new server. If you are editing a server entry, you can edit the name field to rename the server (the new name must not already exist in *interfaces*).
- Available network transports – a list of the network addresses upon which the server accepts client connections. You can edit the list of addresses as follows:
  - Select Add Network Transport or Modify Network Transport to create a new address or edit an existing address. See "Adding or editing network transport addresses," next, for details.

- Clicking Delete Network Transport removes a selected network address.
- If the server entry has multiple addresses, clicking Move Network Transport Up or Move Network Transport Down allows you to rearrange the order of addresses in the list.

- OK – commits your changes and closes the window. Note that changes to *interfaces* are not applied until you close the session.
- Cancel – closes the window and discards any edits.

Adding or editing network transport addresses

The Network Transport Editor allows you to view, edit, or create the transport addresses at which a server accepts client connections. This window displays the name of the server entry for the address and allows you to configure the following items:

- Transport type – specifies the protocol and interface for the address with the value as tcp.
- Address information – depending on the transport type, different address components are required. The sections below discuss address formats in detail.

TCP/IP addresses

Indicate a TCP/IP address by choosing tcp from the Transport type menu. In *interfaces* entries, you must use the tli tcp protocol for:

- Adaptive Server or Replication Server® version 11.0.x or earlier on platforms that use tli-formatted *interfaces* entries.
- Open Client and Open Server® version 12.0 or earlier on platforms that use tli-formatted *interfaces* entries.

Note tli entries in the interfaces file have been deprecated in Open Client and Open Server versions 12.5 and later. SDK and Open Server including DB-Library, supports the tli format, however, Sybase does not recommend its use.

- On Solaris, DB-Library supports tcp format.

For other clients and servers, use the tcp transport type.

The address information for a TCP/IP entry consists of a host name (or IP address) and a port number (entered as a decimal number). For tli tcp-formatted *interfaces* entries, the host's IP address and the port number are converted to the 16-byte hexadecimal representation required for tli tcp-formatted *interfaces* entries.

Troubleshooting dsedit or dsedit problems

This section lists some common problems and describes how to correct them.

dsedit does not start

Check for the following:

- SYBASE environment variable is not set or points to the wrong directory.
- X11 is not configured correctly. If you are running dsedit on a remote host, verify that X11 clients on the remote host can connect to the X11 server on your own machine. See your X11 documentation for more troubleshooting information. If X11 is not available, use dscp instead of dsedit.

Cannot add, modify, or delete server entries

Check for the following:

- Permissions problems with the *interfaces* file

  To edit interfaces entries, you must have write permission on the *interfaces* file and the Sybase installation directory.

APPENDIX A

Environment Variables

This appendix describes environment variables that contain configuration information.

Environment variables used for connection

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

Table A-1: Environment variables used for connection

Variable	Value	Used by
DSLISTEN	The name of the Open Server application, as listed in *interfaces* or directory service. If DSLISTEN is not set, Open Server uses the default value "SYBASE."	Open Server
DSQUERY	The name of the target server, as listed in *interfaces* or directory service. If DSQUERY is not set, Open Client uses the default value "SYBASE."	Open Client
SYBASE	The location of the Sybase home directory. **Note** The CS_SYBASE_HOME property, which specifies the name and path to an alternate Sybase home directory, overrides the environment variable *$SYBASE*.	Open Client
SYBASE_OCS	Home directory for the Open Client and Open Server products.	*$SYBASE/$SYBASE_OCS*

Environment variables used for localization

Note The *LC_xxxx* variables are not used by DB-Library.

Open Client and Open Server products use these environment variables during localization:

- LC_ALL
- LC_COLLATE
- LC_TYPE
- LC_MESSAGE
- LC_TIME

The localization environment variables are POSIX standard environment variables and can be used by non-Sybase applications.

Some non-Sybase applications can use the same localization-related environment variable as your Open Client and Open Server application. Make sure that *locales.dat* lists the same locale names as are used by the environment variables of the non-Sybase applications.

Environment variables used for configuration

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

Table A-2: Environment variables used for configuration

Environment variable	Description	Used during
SYBOCS_CFG	Overrides the *$SYBASE/SYBASE_OCS/config/ocs.cfg* default external configuration file path. For more information, see the *Open Client Client-Library/C Reference Manual*.	Runtime

Environment variable	Description	Used during
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.	Runtime
SYBOCS_TCL_CFG	Sets the full path name of the *libtcl.cfg* and *libtcl64.cfg* files. For example: %setenv SYBOCS_TCL_CFG /usr/u/joe/libtcl.cfg	Runtime

Setting environment variables

This section gives instructions for setting environment variables in the C shell and the Bourne shell.

To set environment variables in the C shell, use this command:

```
setenv VARIABLE value
```

For example, the following command defines the DSQUERY environment variable as “test”:

```
setenv DSQUERY test
```

To set environment variables in the Bourne shell, use this command:

```
VARIABLE=value; export VARIABLE
```

For example, the following command defines the DSQUERY environment variable as “test”:

```
DSQUERY=test; export DSQUERY
```

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* directory structure. Open Client and Open Server products use the configuration files listed in Table B-1.

Table B-1: Names and locations for configuration files

File name	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/config* **Note** Use the CS_LIBTCL_CFG property or the SYBOCS_TCL_CFG environment variable to specify an alternate path to *libtcl.cfg* file.	See "The libtcl.cfg and libtcl64.cfg files" on page 60. Also see the *Open Client Client-Library/C Reference Manual*.
interfaces	The *interfaces* file contains connection and security information for each server listed in the file. It is also used as a backup for services described in the *libtcl.cfg* file.	*$SYBASE*	See "The interfaces file" on page 68.

File name	Description	Location	For more information
objectid.dat	The object identifiers file maps global object identifiers to local names for character set, collating sequence, and security mechanisms.	*$SYBASE/config/objectid.dat*	See Appendix C, "Localization."
ocs.cfg	The runtime configuration file allows you to change certain values at runtime.	*$SYBASE/$SYBASE_OCS/config*	See "The ocs.cfg file" on page 71.

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

The *libtcl.cfg* and the *libtcl64.cfg* files (collectively, *libtcl*.cfg* file) are the driver configuration files that contain information about two 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 support multiple service providers.

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

Dynamic linking of drivers

Client-Library and Server-Library support dynamic loading of directory and security drivers. This allows you to change a driver that an application uses and to use features as they become available at your site, without re-linking the application.

$SYBASE/$SYBASE_OCS/config/libtcl.cfg configures directory and security drivers. This file maps symbolic strings to the appropriate driver and any required initialization information.

Client-Library or Server-Library applications, including Sybase utility programs such as dscp, locate the appropriate drivers specified in *libtcl.cfg* as follows:

1 If the driver file name in *libtcl.cfg* has path components (contains a slash), that path is used. Otherwise, the search continues to step 2.

2 Depending on your platform, the directories specified by these environment variables are searched. If the driver is not located, the search continues to step 3.

The library locations and environment variables are listed in Table 5-5 on page 28.

3 The path *$SYBASE/$SYBASE_OCS/lib* is used (or *$SYBASE/$SYBASE_OCS/devlib* for applications built with debug-mode libraries).

How *libtcl.cfg* is used

Open Client and Open Server read the *libtcl.cfg* file when loading a directory, or security driver. *libtcl.cfg* is located in the *$SYBASE/$SYBASE_OCS/config* directory.

The CS_LIBTCL_CFG configuration property specifies the name and path to an alternate *libtcl.cfg* file.

An entry in *libtcl.cfg* provides Open Client and Open Server products with the name of the driver and its initialization information.

How *libtcl.cfg* is structured

The *libtcl.cfg* file is divided into two sections, one for each type of driver. The sections are titled:

- [DIRECTORY]
- [SECURITY]

To use Open Client and Open Server, directory or security service support, you must use the appropriate software to support these services.

DIRECTORY section

The [DIRECTORY] section lists directory drivers. The syntax for a directory driver entry is:

```
provider=driver init-string
```

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 driver. The default location of all drivers is in *$SYBASE/$SYBASE_OCS/lib*. The LDAP directory driver is platform dependent, for example:
  - *libsybdldap.sl* for HP HP-UX PA-RISC.
  - *libsybdldap.so* for HP HP-UX Itanium, IBM AIX POWER, Solaris platforms, and Linux platforms.
- *init-string* is an initialization string for the driver. The value for *init-string* varies by driver.

For LDAP entries in the DIRECTORY section

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

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

where the *ldapurl* is defined as follows:

```
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.so 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:

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

To set the password:

```
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. The pwdcrypt utility is located in *$SYBASE/$SYBASE_OCS/bin*.

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

```
bin/pwdcrypt
```

When prompted, enter your password twice.

The pwdcrypt utility generates an encrypted password, for example:

```
0x01312a775ab9d5c71f99f05f7712d2cded2i8d0ae1ce78868d0e8669313d1bc4c706
```

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

Note The LDAP URL must be on a single line.

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

Replace the password with the encrypted string:

```
ldap=libsybdldap.so
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 section

The [SECURITY] section lists security drivers. The syntax for a security driver entry is:

```
provider=driver init-string
```

where:

- *provider* is the local name for the security mechanism. The local name of the security mechanism is listed in the object identifiers file, *$SYBASE/config/objectid.dat*.

  See “The objectid.dat file” on page 80 for information about *objectid.dat*.

  The default local name for the Kerberos security mechanism is “csfkrb5.” If you use a local mechanism name other than the default, you must add an alias for the name in the object identifiers file, after the default name. (See “An objectid.dat example” on page 81 for an example.)

- *driver* is the name of the driver. The default location of all drivers is in *$SYBASE/$SYBASE_OCS/lib*.

Table B-2 lists the supported security drivers for each platform:

Table B-2: Supported security drivers

Platform	Security type	Security driver	Service compatibilities
HP HP-UX PA-RISC 32-bit	Kerberos	*libsybskrb.sl*	CyberSafe TrustBroker 2.1 MIT Kerberos 1.4.1
HP HP-UX PA-RISC 64-bit	Kerberos	*libsybskrb64.sl*	MIT Kerberos 1.4.3
HP HP-UX Itanium 32-bit	Kerberos	*libsybskrb.so*	MIT Kerberos 1.4.1
HP HP-UX Itanium 64-bit	Kerberos	*libsybskrb64.so*	MIT Kerberos 1.4.1
IBM AIX POWER 32-bit	Kerberos	*libsybskrb.so*	CyberSafe TrustBroker 2.1 MIT Kerberos 1.4.1
IBM AIX POWER 64-bit	Kerberos	*libsybskrb64.so*	CyberSafe TrustBroker 2.1 MIT Kerberos 1.4.3
Linux x86 32-bit	Kerberos	*libsybskrb.so*	MIT Kerberos 1.4.1
Linux x86-64 64-bit	Kerberos	*libsybskrb64.so*	MIT Kerberos 1.4.1
Linux POWER 32-bit	Kerberos	*libsybskrb.so*	MIT Kerberos 1.4.1
Linux POWER 64-bit	Kerberos	*libsybskrb64.so*	MIT Kerberos 1.4.1
Solaris x86-64 32-bit	Kerberos	*libsybskrb.so*	MIT Kerberos 1.4.2
Solaris x86-64 64-bit	Kerberos	*libsybskrb64.so*	MIT Kerberos 1.4.2
Solaris SPARC 32-bit	Kerberos	*libsybskrb.so*	CyberSafe TrustBroker 2.1 MIT Kerberos 1.4.1
Solaris SPARC 64-bit	Kerberos	*libsybskrb64.so*	CyberSafe TrustBroker 2.1 MIT Kerberos 1.4.1

- *init-string* is an initialization string for the driver. Its value varies by driver.

  For the Kerberos driver, the syntax for *init-string* is:

  ```
  secbase=@realm [libgss=<gss api V1 compatible
  library>]
  ```

  where:

  - *realm* is the default Kerberos realm name.

- (Optional) *libgss* is the full path to a GSS API version 1 compliant library.

The following [SECURITY] sections show entries for CyberSafe Kerberos drivers on Solaris:

- Kerberos

```
[SECURITY]
 csfkrb5=libsybskrb.so secbase=@ASE libgss=/krb5/lib/libgss.so
```

where libgss=*/krb5/lib/libgss.so*, which means that the default Kerberos realm is Adaptive Server, and that the GSS library to load is */krb5/lib/libgss.so*

Note Be aware that the `libgss=<gss shared object path>` that specifies the GSS API library is to be used. It is important that you distinctly locate the Kerberos Client libraries being used, especially when multiple versions are installed on a machine.

Adding a directory driver

❖ **Adding a directory driver to *libtcl.cfg***

1 Choose a value for *provider*, which can have any value.

Note To make an entry the default directory driver, add it as the first entry in the DIRECTORY section.

2 Determine the value of *driver*, which varies by platform:

- For IBM AIX POWER, Solaris and Linux platforms, and HP HP-UX Itanium use *libsybdldap.so*.
- For HP HP-UX PA-RISC use *libsybdldap.sl*.

3 Verify the host and port number of the LDAP server.

4 Determine the value of the DIT base, which is the location where LDAP begins its search for the server entry.

5 Ensure that the DIT base path exists in the LDAP directory.

The LDAP administrator may need to perform this task. See your LDAP documentation for more information.

6 Go to the [DIRECTORY] section and add an entry using the following format:

```
provider=driver ldap://host:port/ditbase
```

The following is an example for an LDAP driver:

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

You can add two or more LDAP driver entries that use different DIT bases. Multiple driver entries are useful when you want to use the dscp or dsedit tools to view and modify entries that are in different locations in the LDAP directory. For example, you might add the entries below:

```
[DIRECTORY]
ldap=libsybdldap.so ldap://lserv:389/dc=production,dc=sybase,dc=com
ldap1=libsybdldap.so ldap://lserv:389/dc=test,dc=sybase,dc=com
ldap2=libsybdldap.so ldap://backup1:389/dc=sybase,dc=com
```

Adding a security driver

❖ **Adding a security driver to *libtcl.cfg*:**

1 Determine the value of *provider*, which is the local name of the security mechanism, as listed in the object identifiers file, *$SYBASE/config/objectid.dat*. The default local name for Kerberos is *csfkrb5*.

2 Determine the value of *driver*, which varies by platform and security mechanism. (Table B-2 on page 65 lists driver names.)

3 Determine the value of *init-string*.

For the Kerberos driver, *init-string* has the following form:

```
secbase=@realmname [libgss=<gss api V1 compatible
library>]
```

where:

- *realmname* is the default realm name for unqualified CyberSafe user names.
- (Optional) *libgss* is the full path to a GSS API version 1 compliant library.

4 Go to the [SECURITY] section and add an entry using the following format:

```
provider=driver init-string
```

For example:

```
csfkrb5=libsybskrb.so secbase=@ASE
libgss=/krb5/lib/libgss.so
```

The *interfaces* file

The *interfaces* file contains information about the network locations of servers.

Open Client and Open Server use *interfaces* as a limited-function directory service. The *interfaces* file also serves as a default if an external directory service fails.

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

The *interfaces* file is created during installation as *$SYBASE*/interfaces. Open Client and Open Server products look for *interfaces* in *$SYBASE*.

An application can look for *interfaces* in a location other than the default location. For more information, see ct_config in the *Open Client Client-Library/C Reference Manual* and srv_props in the *Open Server Server-Library/C Reference Manual*.

interfaces entries

Open Client and Open Server use a standard format for *interfaces* entries.

Standard format

An *interfaces* entry has the following form:

```
# put comments here<newline>
 SERVERNAME[<tab>retry_count<tab>retry_delay]<newline>
 <tab>{master|query} protocol network host
port<newline>
 <tab>[secmech mechanism1,..., mechanismn]<newline>
 <blank line>
```

where:

- *SERVERNAME* is an alias by which Open Client and Open Server recognize which *interfaces* entry to read. *SERVERNAME* must begin with a letter (ASCII a-z, A-Z), contain letters, numbers, and underscores only, and have a maximum of 11 characters.
- *retry_count* (optional) determines the number of times a client tries to connect to a server after an initial failure to connect.
- *retry_delay* (optional) determines the time interval between connection attempts.
- "master | query" specifies the type of connection:
  - "master" specifies a master line, which is used by server applications to listen for client queries.
  - "query" specifies a query line, which is used by client applications to find servers.

  The master line and the query line of an *interfaces* entry contain identical information. The dscp utility creates both types of lines for each entry. The resulting entry can be used by both clients and servers.
- *protocol* is the name of the network protocol. Valid value is "tcp" for TCP/IP.
- *network* is a descriptor of the network.

  Open Client and Open Server do not currently use *network*; it is a placeholder should Sybase need to define this information in the future.
- *host* is the network name of the node, or machine, that the server is running on. The maximum number of characters for *host* depends on the protocol specified in the entry. For TCP/IP, the maximum is 32.

  Use the /bin/hostname command to determine the network name of the machine you are logged in to.
- *port* is the port used by the server to receive queries. The registered TCP/IP port numbers range from 1024 to 49151. Sybase recommends to use a port number from this range.

  Use the netstat command to check which port numbers are in use.
- The optional SECMECH line contains the identifier used to list the security mechanisms that a server supports.
- *mechanism1,..., mechanismn* are the security mechanisms that a server supports. You can specify multiple security mechanisms by using a comma 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 80 for more information about object identifiers.

Editing the *interfaces* file

Edit *interfaces* with dscp or an operating system editor, such as vi.

Using dscp to edit an *interfaces* file makes the process easier, because it correctly formats the address string that you enter. See Chapter 7, "Using dscp," for complete instructions about editing an *interfaces* file using dscp.

Standby server addressing

You can set up your *interfaces* file to allow for *standby server addressing*, which allows Open Client to connect with an alternate server if the first connection attempt fails.

For example, the following *interfaces* entry directs the application to the server at port number 1025 on the machine named "violet." If this server is not available, the connection fails.

```
#
 BETA
     query tcp hp-ether violet 1025
     master tcp hp-ether violet 1025
     secmech 1.3.6.1.4.1.897.4.6.1
```

However, if the BETA entry has multiple *query* lines, Open Client automatically attempts to connect to the next server listed when the first connection attempt fails. Such an *interfaces* entry might appear as follows:

```
#
 BETA
 query tcp hp-ether violet 1025
 query tcp hp-ether plum 1050
 query tcp hp-ether mauve 1060
 master tcp hp-ether violet 1025
```

```
secmech 1.3.6.1.4.1.897.4.6.1
```

Note The *SERVERNAME* element of an *interfaces* entry is an *alias* and does not uniquely identify the actual server. The host and port elements uniquely identify the server.

In the previous example, if Open Client fails to connect to "violet" at port 1025, Open Client attempts to connect to the server listed in the next query line, called "plum," at port 1050, and so on.

Any number of alternate servers may be listed under a server's *interfaces* entry, but each alternate server must be listed in the same *interfaces* file.

The ocs.cfg file

The runtime configuration file *ocs.cfg* 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; therefore, the application's settings can be changed without recompiling the code.

Client-Library does not read *ocs.cfg* by default, but all *Client-Library*-based applications attempt to read the file if the file name exists in *$SYBASE/$SYBASE_OCS/config*. 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 that can be set in the file.

APPENDIX C

Localization

Localization is the process of initializing an application so that it executes 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 ways:

- Using initial localization values
- Using both 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 a need exists 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*. Open Client and Open Server always 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 environment variables shown in Table C-1.

Table C-1: Environment variables used for localization

Environment variable	Description	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

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

Before running a localized application:

- Verify that *locales.dat* contains an entry which reflects the localization values the application uses. If it does not, add an appropriate entry.
- Verify that the localization files that your application uses are installed:
  - Localized message files are located in the *$SYBASE/locales/message* directory.
  - Collating sequence files are located in the *$SYBASE/charsets* directory.

  All Open Client and Open Server products include files to support at least one language and one or more character sets and collating sequences (sort orders). During installation, these files are loaded into the *$SYBASE* directory structure in the appropriate locations. When configuring an Open Client or Open Server application, you must verify that the previous directories contain the correct files for your site and application.

Localization files

At runtime, Open Client and Open Server applications load localization information from external files. Three directories in the *$SYBASE* 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 versions of Open Client and Open Server software. These directories contain localized message files organized by character set.
  - *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 *config* directory contains:

- The *objectid.dat file*, which maps global names for objects such as character sets and languages to local platform-specific names.

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

The locales file, called *locales.dat*, provides platform-specific locale information in a Sybase proprietary format. This file associates locale names with languages, character sets and collating sequences.

How it is used

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

Location of *locales.dat*

The *locales.dat* file is located in the *$SYBASE*/*locales* directory. See “Localization files” on page 75 for a diagram of the *$SYBASE*/*locales* directory structure.

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 *locales.dat* list POSIX values for locale names.
- , (comma) is the list separator character for the file.
- *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 file sample

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

```
[aix]

      locale = C, us_english, iso_1
      locale = En_US, us_english, iso_1
      locale = en_US, us_english, iso_1
      locale = default, us_english, iso_1
      locale = japanese.sjis, japanese, sjis
      locale = japanese, japanese, eucjis
      locale = us_english.utf8, us_english, utf8
```

Editing *locales.dat*

If the predefined entries in *locales.dat* do not meet your needs, edit the file with an operating system editor such as vi.

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.

Edit *locales.dat* to:

- 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 any 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, type in or change the appropriate entry.

   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.

Note It is not necessary to 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 *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 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* entries. If a *locales.dat* entry does not specify a collating sequence, then a binary sort order is used with the locale.

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 *config* directory

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

The *objectid.dat* file

The *objectid.dat* file, which is located in the *$SYBASE/config* directory, 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. It is based on a naming tree defined by the international standards bodies CCITT and ISO.

objectid.dat sections and entries

The *objectid.dat* file 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.

An *objectid.dat* example

The following sample 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]
     1.3.6.1.4.1.897.4.6.3 = NTLM
     1.3.6.1.4.1.897.4.6.6 = csfkrb5
```

Editing *objectid.dat*

Edit *objectid.dat* with an operating system editor such as vi if you change the local name of an object.

APPENDIX D

Kerberos Security Services

This appendix lists the security services supported by the Kerberos security driver and summarizes some system configuration tasks that are required in order to use the Kerberos security driver.

Note DB-Library does not support Kerberos.

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

Supported 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

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

Configuring CyberSafe Kerberos

- Install CyberSafe GSS Run Time Library.
- Set the credentials (desired security features) using ct_con_props, or use the default credentials by not setting credential properties.
- Configure the security section of the *libtcl.cfg* or *libtcl64.cfg* configuration file.
- 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 before running the client application.
- Log in to the CyberSafe security mechanism with the CyberSafe utility kinit before running your Client-Library 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 CyberSafe credential.
- The environment variable CSFC5CCNAME, 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 CyberSafe documentation.
- The *libgss.so* or *libgss.sl* file must be in the path while running your Client-Library application. This file is not provided by Sybase, but is included with some CyberSafe products. If this file is not included with your CyberSafe product, contact CyberSafe to obtain their GSS-API library.
- 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, you can use isql to test your configuration.

See *README.SEC* in the *$SYBASE/$SYBASE_OCS/sample/srvlibrary* directory for an example of configuring and running the sample program.

Open Server applications and CyberSafe Kerberos

You can run a custom Open Server application or the Security Guardian server with CyberSafe Kerberos security. In order for the server and its clients to communicate over the network, you must perform the normal configuration steps described in Chapter 3, "Basic Configuration for Open Server." Then, for the server and its clients to use CyberSafe Kerberos security services, you must perform these additional configuration steps:

1 Decide which CyberSafe Kerberos principal the server will run as.

  You can create a new principal with the CyberSafe kadmin utility, using the add command. The principal must be allowed to act as a server.

2 If the server principal does not already have a key in a CyberSafe Kerberos server key table file, create one with the CyberSafe kadmin utility, using the ext command. Make sure that the operating system user who starts the server has read permission on the server key table file. In a production environment, you must control the access to the key table file. If a user can read this file, they can create a server that impersonates your server.

3 Make sure the CyberSafe Kerberos security driver is configured in the [SECURITY] section of *libtcl.cfg*. See "SECURITY section" on page 64 for details.

4 Set the CSFC5KTNAME environment variable to the name of the key table file that holds the key for the server principal (see step 2). The CyberSafe runtime libraries require that this environment variable to be set if the server key table file is in a location other than the CyberSafe system default.

5 You must place the shared library file (*libgss.so* on Solaris and Linux platforms, *libgss.so* on IBM AIX POWER, or *libgss.sl* on HP HP-UX) in a directory specified in the shared library path for your platform (see Table 5-5 on page 28). As an alternative, you can use the *libgss* keyword in *libtcl.cfg* to specify the path to the *GSS* library.

  What enables the client to find this shared library file at runtime. You can also place the shared library file in the *lib* subdirectory of the CyberSafe installation as long as this subdirectory is in the shared library path.

  This shared library is not provided by Sybase, but it is included in some CyberSafe products. If it is not included with your CyberSafe product, contact CyberSafe to obtain their GSS-API library.

6 When you start the server, specify the principal name in addition to the network name if the principal name does not match the network name. You do not have to specify the network name if you set the DSLISTEN environment variable to the network name.

The Open Server's network name is its name in *interfaces* or the directory service.

A custom Open Server application specifies the principal name by setting the SRV_S_SEC_PRINCIPAL Server-Library property.

Kerberos does not allow the *key table* file to be specified programatically; you must use the CSFC5KTNAME environment variable (see step 4).

Client-Library applications and CyberSafe Kerberos

See "Client-Library and security services" on page 34 for an overview of how client applications use security services. These considerations apply to client applications that use CyberSafe Kerberos security services:

- The application must use a preexisting user credential to connect to the server. In other words, the user of the application must log in to CyberSafe before running the client application. On UNIX, use the CyberSafe kinit utility to log in to CyberSafe.
- 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 credential.

Configuring MIT Kerberos

- Install and configure the MIT software on your system. Refer to Table B-2 on page 65 to see which MIT version is supported on your platform.
- Set the desired security features using ct_con_props, or use the default credentials by not setting credential properties.
- Configure the security section of the *libtcl.cfg* configuration file.

- 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, *libgssapi_krb5.so*, 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.

See *README.SEC* in the *$SYBASE_OCS/sample/srvlibrary* directory for an example of configuring and running the sample program.

Open Server applications and MIT Kerberos

You can run a custom Open Server application with Kerberos security. In order for the server and its clients to communicate over the network, you must perform the normal configuration steps described in Chapter 3, "Basic Configuration for Open Server." In order for the server and its clients to use Kerberos security services, you must perform these additional configuration steps:

1 Decide which Kerberos principal the server will run as.

  You can create a new principal with the kadmin utility, using the add command. The principal must be allowed to act as a server.

2 If the server principal does not already have a key in a Kerberos server key table file, create one with the kadmin utility, using the ext command. Make sure that the operating system user that starts the server has read permission on the server key table file. In a production environment, you must control the access to the key table file. If a user can read this file, they can create a server that impersonates your server.

3 Make sure the Kerberos security driver is configured in the [SECURITY] section of *libtcl.cfg*. See "SECURITY section" on page 64 for details.

4 Set the KRB5_KTNAME environment variable to the name of the key table file that holds the key for the server principal (see step 2). The Kerberos runtime libraries require this environment variable to be set if the server key table file is in a location other than the system default.

5 Enter the location of *libgssapi_krb5.so* file in the *libtcl.cfg* directory using the libgss keyword.

6 When you start the server, specify the principal name in addition to the network name if the principal name does not match the network name. You do not have to specify the network name if you set the DSLISTEN environment variable to the network name.

   The Open Server network name is defined in the *interfaces* directory service.

   A custom Open Server application specifies the principal name by setting the SRV_S_SEC_PRINCIPAL Server-Library property.

   Kerberos does not allow the *key table* file to be specified programatically; you must use the KRB5_KTNAME environment variable (see item 4).

Client-Library applications and MIT Kerberos

See "Client-Library and security services" on page 34 for an overview of how client applications use security services. These considerations apply to client applications that use Kerberos security services:

- The application must use a preexisting user credential to connect to the server. In other words, the user of the application must log in to Kerberos before running the client application. On UNIX, use the Kerberos kinit utility to log in to Kerberos.
- 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 Kerberos credential.

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 isql and bcp option -Vd. For more information, see the *Open Client and Open Server Programmers Supplement for UNIX*.

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*.

Configuring Solaris Kerberos

Solaris Kerberos is based on MIT's Kerberos with the following differences:

- The GSS library is */usr/lib/libgss.so* instead of *libgssapi_krb5.so*.
- All other information in this section, Configuring MIT Kerberos applies to the version of Kerberos provided with Solaris.

Configuring Kerberos environments and mixed Kerberos environments

For suggestions about configuring the Kerberos environment and mixed Kerberos environments, refer to the technical document General Kerberos Configuration Tasks at http://www.sybase.com/detail?id=1029260.

APPENDIX E

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 in order 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 description

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,” described next.

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

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

SSL security levels in Open Client and Open Server

SSL provides several levels of security:

- 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.

The SSL filter

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 *interfaces* 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 Kerberos, that are defined with secmech (security mechanism) lines in the *interfaces* file. The master and query lines determine the security protocols that are enforced for the connection.

For example, a typical *interfaces* file on a UNIX machine using SSL looks like this:

```
SERVER <retries><time-outs>

  master tcp ether <hostname> <portnumber> ssl
  query tcp ether <hostname> <portnumber> ssl
```

where *hostname* is the name of the server to which the client is connecting and *portnumber* is the port number of the host machine.

All connection attempts to a master or query entry in the *interfaces* 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 *interfaces* file on UNIX that supports both SSL-based connections and plain-text connections looks like this:

```
SYBSRV1
    master tcp ether hostname 2748 ssl
    query tcp ether hostname 2748 ssl
    master tcp ether 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 Certificate of Authority (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" next.
- For connections to SSL-enabled servers, the common name in the server’s certificate must match the server name in the *interfaces* 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 the following directory, *$SYBASE/$SYBASE_ASE/certificates/servername.crt*, where *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.

Syntax for UNIX

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

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

ASESSL1
```

```
master tcp ether hostname1 5000 ssl="CN=name1"
query tcp ether hostname1 5000 ssl="CN=name1"

ASESSL2
master tcp ether hostname2 5000 ssl="CN=name2"
query tcp ether hostname2 5000 ssl="CN=name2"

ASESSL3
master tcp ether hostname3 5000 ssl="CN=name3"
query tcp ether hostname3 5000 ssl="CN=name3"

ASESSL4
master tcp ether hostname1 5000 ssl="CN=name4"
query tcp ether hostname1 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/config/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,
  "$SYBASE/config/trusted.txt", CS_NULLTERM, NULL);
  ```

  where *$SYBASE* 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,
  "$SYBASE/config/trusted.txt", CS_NULLTERM, NULL);
  ```

where *$SYBASE* 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 server 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 “The certpk12 utility” on page 104.

To test the certificate request tool and to verify that the authentication methods are working on your server, Open Client and Open Server provides a certreq and certauth tool, for testing purposes, that allows 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 request certificates

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 "The certpk12 utility" on page 104.

Using Sybase tools to request and authorize certificates

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

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

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.

See the following section for a description of these Sybase tools.

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.

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

Description of Sybase tools

The following sections describe the Sybase tools you can use to request certificates.

The certauth utility

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.

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.

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 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 upon 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

The certreq utility

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 94 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, and is therefore in interactive mode. To create a server certificate request (*server_req.txt*) and its private key (*server_pkey.txt*), enter the following:

```
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

The certpk12 utility

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 example 1 and 2, *caRSA.crt* and *caRSA_new.crt* are identical. However, *caRSApkey.txt* and *caRSApkey_new.txt* are different because they are encrypted randomly.

Example 3

This example exports the certificate file called *caRSA.crt* to a PKCS#12 file called *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 called *caRSAcert.p12*, which contains a certificate. It outputs the embedded certificate to a text file called *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. UNIX platforms that support this feature require the Certicom Security Builder shared library named *libsbgse2.so*, which is installed with CSI 2.6. To support FIPS encryption, a Certicom Security Builder shared library named *libsbgse2.so* is installed in *$SYBASE/$SYBASE_OCS/lib3p* or in *$SYBASE/$SYBASE_OCS/lib3p64* when you install SDK or Open Server

Index

E

G

H

I

K

L

M

N

O

P

R

S

T

U

V