SYBASE®

Configuration Guide

Open Client™

15.0

[Mac OS X]

DOCUMENT ID: DC00965-01-1500-01

LAST REVISED: October 2008

Contents

About This Book

The Open Client *Configuration Guide* for Mac OS X contains information about configuring your system to run Open Client™ products on Mac OS X 10.5 or later on Intel, 32-bit.

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 *Configuration Guide* for Mac OS X 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, "Configuring Open Client for Sybase Failover," describes the steps necessary to configure your Open Client applications to connect to the secondary server during failover.

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 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, “Secure Sockets Layer in Open Client and Open Server” describes the Secure Sockets Layer (SSL) support for Open Client and summarizes some system configuration tasks that are required in order to use the SSL protocol.

Related documents

- The Open Server and SDK *New Features* for Microsoft Windows, Linux, UNIX, and Mac OS X 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 Software Developer’s Kit *Release Bulletin* contains last-minute information about the release.
- *Installing Sybase Products* contains installation procedures for installing your Open Client software.
- The Open Client *Client-Library/C Reference Manual* contains reference information for Open Client Client-Library.
- The Open Client *DB-Library/C Reference Manual* contains reference information for DB-Library™.
- The Open Client *Client-Library/C Programmer’s Guide* contains information on how to design and implement Client-Library programs.
- 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 Client *Programmer’s Supplement* for Mac OS X contains information for programmers using Open Client products. This document includes information about:
    - Compiling and linking an application
    - The sample programs that are included online with Open Client products

    - Routines that have platform-specific behaviors

- The Adaptive Server (called “SQL Server” in versions prior to 11.5) *Reference Manual* describes Sybase Adaptive Server® Enterprise commands, datatypes, functions, and system procedures.
- The *Transact-SQL User’s Guide* documents Transact-SQL®, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. 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 SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals 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 Certification Report.

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

4 Click a 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 manuals 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, which is packaged and is part of the Software Developer's Kit (SDK), according to the instructions in the *Installation Guide* for UNIX.

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

About Open Client

Open Client provides three application programming interfaces (APIs), named *dblib* and *ctlib*, and Bulk-Library. These products enable communications between Adaptive Server® (called "SQL Server in versions prior to 11.5) and Open Server applications and customer applications, third-party products, and other Sybase products.

See the following documents for detailed information about Open Client:

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

Overview of configuration

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

Open Client uses configuration information to:

- Initialize the Open Client (except for DB-Library) application
- Establish a connection with Adaptive 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 on DB-Library, see the Open Client *DB-Library/C Reference Manual*.

The initialization process

To initialize an application, Open Client:

- 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 [FILTER] section of the *libtcl.cfg* file to specify the SSL security mechanism.

Note LDAP and Kerberos are not supported on Mac OS X on Intel, 32-bit. Your connection will fail if you use *libtcl.cfg* to load these directory and security drivers.

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 the *interfaces* file to obtain the target server's address.

Configuration tasks

You must complete some basic configuration tasks for an Open Client 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 applications do not explicitly specify a name of a server.
- Ensuring that the target server's address is available.
- Configuring your network driver, if needed.

There are additional tasks if you plan to customize the localization values in addition to or instead of using the 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:

Note Items 1-3 (do not apply to DB-Library) 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.

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

2 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 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 *interfaces*. DB-Library does not examine the *libtcl.cfg* file, it accesses the *interfaces* file:

   - *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.

Adaptive Server version 15.0 can store data that has different limits than data stored in previous versions. The Client-Library clients can use the new limits, providing the version is set to CS_VERSION_150 at init time, but the DB-Library clients cannot. Clients also must be able to handle the new limits the data can use. Open Client version 15.0 supports Adaptive Server 15.0 limits. If you are using earlier versions of Open Client, they cannot process the data if you:

- Upgrade to Adaptive Server version 15.0

- Drop and recreate the tables with wide columns
- Insert wide data

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. Configure *interfaces*:

   Create a server entry in *interfaces* service using an editor of your choice.

   See "The interfaces file" on page 18 for information about *interfaces*.

Client-Library compatibility

Client-Library on Mac OS X is certified to work with Adaptive Server and Open Server products shown in Table 2-1.

Table 2-1: Open Client compatibility

Platform	**Open Client 15.0**	**Open Client 12.5.1**	**Open Server 12.5.1**	**Adaptive Server 12.5**
Mac OS X 10.4 on PowerPC, 32-bit	n/a	x	x	x
Mac OS X 10.5 on Intel, 32-bit	x	n/a	n/a	n/a

LEGEND: x = compatible; n/a = product not available on that platform.

In addition, note these compatibility issues for Open Client/C:

- The dynamic libraries must be the same version level as the libraries used to build the application.
- Header files included in an application must be the same version level as the library with which the application is linked.

CHAPTER 3

Configuring Open Client for Sybase Failover

The Sybase Failover feature is documented in *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.

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.

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
```

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
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.	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 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 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 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
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

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 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 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 required initialization information.	*$SYBASE/$SYBASE_OCS/config*	See "The libtcl.cfg file" on page 18.
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.	*$SYBASE*	See "The interfaces file" on page 18.
objectid.dat	The object identifiers file maps global object identifiers to local names for character set, collating sequence, and security mechanisms.	*$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 21.

The *libtcl.cfg* file

Open Client reads the [FILTER] entry in *libtcl.cfg* file when loading customized SSL functionality. *libtcl.cfg* is located in the *$SYBASE/$SYBASE_OCS/config* directory.

The *interfaces* file

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

Open Client uses *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.

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

interfaces entries

Open Client version 11.1 and later 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 recognizes 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.
- *protocol* is the name of the network protocol. Valid values are:
  - "tcp" for TCP/IP – all UNIX platforms
  - "decnet" for DECnet
- *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.
  - For DECnet, the maximum is 6.

  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 TCP/IP and DECnet protocols specify this element differently:
  - TCP/IP: Registered port numbers range from 1024 to 49151. Sybase recommends to use a port number from this range.

- DECnet: Valid object numbers range from 128 to 253. Object names are also valid.

  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 30 for more information about object identifiers.

Editing the *interfaces* file

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

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 Developer's Guide*.

Overview of the localization process

Open Client applications can localize in two ways:

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

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

In addition, an Open Client 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 Clientuses environment variables to determine which locale name to look for in *locales.dat*. Open Client always search for the following environment variables:

- LC_ALL
- LANG, if LC_ALL is not set

When setting up custom localization values, Open Client 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 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 Developer's 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 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 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, organized by language name.
    - *language_name* subdirectories, which are included to provide compatibility with previous versions of Open Client 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 applications use *locales.dat* to determine what localization information to load. The *locales.dat* file directs Open Client 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 *$SYBASE*/*locales* directory. See "Localization files" on page 25 for a diagram of the *$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 applications to generate messages in a variety of languages.

All Open Client 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
- *blklib.loc* – Bulk Library messages
- *bcp.loc* – Bulk Copy messages

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

Open Client 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 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 Developer's 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

Secure Sockets Layer in Open Client and Open Server

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

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 supports, see the Open Client *Client Library/C Reference Manual*.

SSL security levels in Open Client

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 protocol layer that sits on top of the TCP/IP connection.

The SSL filter is different from other security mechanisms, 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 Mac OS X on Intel, 32-bit 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 Mac OS X on Intel, 32-bit 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 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 a SDC environment

The default behavior for SSL validation in Open Client 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.

With the SSL enhancement, Open Client can now support common name validation in an SDC environment. This enhancement allows the ASE 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.

New syntax for UNIX

This is the new syntax of the server entries for the SSL-enabled ASE 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 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*.

Open Client allows you to specify an alternate location for the trusted roots file:

```
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().

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.

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 . For more information about certpk12, see Open Client and Open Server *Configuration Guide* for UNIX

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*.

Note certpk12 is not supported on Mac OS X on Intel, 32-bit. You must use a platform that supports certpk12 to generate a certificate file.

Customized Open SSL support

APPLE Mac OS X on Intel, 32-bit supports SSL functionality using Open SSL.

To enable the SSL functionality, add the *libsybfcsissl.dylib.15.0.7* runtime library to the [FILTER] section of the *libtcl.cfg* configuration file. The configuration file is available in *$SYBASE/$SYBASE_OCS/config*.

Index