SYBASE®

International Developers Guide

Open Client™ and Open Server™

15.5

DOCUMENT ID: DC30525-01-1550-01

LAST REVISED: November 2009

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

Contents

About This Book

Audience

This book is written for Open Client and Open Server application developers. Readers are expected to have a basic knowledge of Client-Library™, DB-Library™, Embedded SQL™, or Server-Library.

How to use this book

This book contains these chapters:

- Chapter 1, “Understanding Internationalization and Localization,” defines internationalization and localization and discusses the advantages of writing international applications.
- Chapter 2, “How Localization Works,” explains how the Open Client and Open Server localization mechanism works.
- Chapter 3, “Writing Internationalized Open Client and Open Server Applications,” explains how to write international Open Client and Open Server applications.
- Chapter 4, “Coded Character Set Conversion Support,” explains how character set conversion works in Open Client and Open Server products.
- Chapter 5, “Editing the Locales File,” describes what is in the locales file and explains how to change it.
- Chapter 6, “Creating or Changing Collating Sequences,” explains how to create and change collating sequence files.
- Appendix A, “Directories and Files Related to Internationalization,” describes the Open Client and Open Server directories and files that are related to internationalization.
- Appendix B, “External Localization File Syntax,” describes external localization file syntax.

Related documents

You can see these books for more information:

- The *Open Server Release Bulletin for Microsoft Windows* contains important last-minute information about Open Server.

- The *Software Developer's Kit Release Bulletin for Microsoft Windows* contains important last-minute information about Open Client™ and SDK.
- The *jConnect™ for JDBC™ Release Bulletin* versions 6.05 and 7.0 contains important last-minute information about jConnect.
- The *Open Client and Open Server Configuration Guide for Microsoft Windows* contains information about configuring your system to run Open Client and Open Server.
- The *Open Client Client-Library/C Reference Manual* contains reference information for Open Client Client-Library™.
- The *Open Client Client-Library/C Programmers Guide* contains information on how to design and implement Client-Library applications.
- 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 Client and Open Server Programmers Supplement for Microsoft Windows* 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 *jConnect for JDBC Installation Guide* version 6.05 contains installation instructions for jConnect for JDBC.
- The *jConnect for JDBC Programmers Reference* describes the jConnect for JDBC product and explains how to access data stored in relational database management systems.
- The *Adaptive Server® Enterprise ADO.NET Data Provider Users Guide* provides information on how to access data in Adaptive Server using any language supported by .NET, such as C#, Visual Basic .NET, C++ with managed extension, and J#.

- The *Adaptive Server Enterprise ODBC Driver by Sybase Users Guide* for Windows and Linux, provides information on how to access data from Adaptive Server on Microsoft Windows, Linux, and Apple Mac OS X platforms, using the Open Database Connectivity (ODBC) Driver.
- The *Adaptive Server Enterprise OLE DB Provider by Sybase Users Guide for Microsoft Windows* provides information on how to access data from Adaptive Server on Microsoft Windows platforms, using the Adaptive Server OLE DB Provider.

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 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 the braces in the command.
[ ]	Brackets mean choosing one or more of the enclosed items is optional. Do not include the braces in the command.
( )	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

Understanding Internationalization and Localization

This chapter defines internationalization and localization and discusses the advantages of writing internationalized applications.

This chapter covers the following topics:

Internationalization and localization

Internationalization is the process of enabling an application to support multiple languages and cultural conventions.

An internationalized application uses external files to provide language-specific information at execution time. Because it contains no language-specific code, an internationalized application can be deployed in any native language environment without code changes.

Localization is the process of configuring an application to execute using a specific language and related cultural conventions (such as datetime representation).

A localized application adopts the look and feel of the native language environment in which it is deployed. It generates messages in the local language and character set and uses local conventions for dates and times.

Open Client and Open Server products provide flexible, powerful localization mechanisms that enable application programmers to design and write internationalized applications.

Advantages of internationalized applications

The task of designing an application to work outside its country of origin can seem daunting. Often, programmers think that internationalizing means hard-coding dependencies based on cultural and linguistic conventions.

A better approach is to write an internationalized application, that is, one that examines the local computing environment to determine what language to use and loads files containing language-specific information at runtime.

When you use an internationalized application, a single application can be deployed in all countries. This has several advantages:

- You write and maintain one application, not half a dozen (or more).
- The application can be deployed, without change, in new countries as needed. You need only supply the correct localization files.
- All sites can expect standard features and behavior.

International systems

An international system may include internationalized client applications, gateways, and servers running on different platforms in different native language environments.

For example, an international system might include the following components:

- Order processing applications in New York City, Mexico City, and Paris (Client-Library applications)
- An inventory control server in Germany (Adaptive Server® Enterprise)
- An order fulfillment server in France (Adaptive Server Enterprise)
- A central accounting application in Japan (an Open Server application working with an Adaptive Server Enterprise)

In this system, the order processing applications:

- Query the inventory control server to determine if requested items are in stock
- Place orders with the order fulfillment server
- Send financial information to the accounting application

The inventory control server and the order fulfillment server respond to queries, and the accounting application collects financial data and generates reports.

All applications and servers use the local language and character set to accept input and generate messages.

In this system, the order processing applications and the Open Server gateway are localized by means of the LC_ALL environment variable, which specifies a locale name. At runtime, Open Client and Open Server applications match the specified locale name to an entry in the Sybase locales file to determine what language, character set, and collating sequence files to load.

The Adaptive Server Enterprises in this system are localized by means of language modules that are installed along with the server.

Open Client and Open Server support for international systems

Open Client and Open Server products provide functionality to fully support the development of international systems. Using Client-Library, Server-Library, and CS-Library, an application can be localized on any supported platform to use:

- A specific language and character set for error messages
- A specific character set when converting strings from another character set
- A specific collating sequence to use when sorting or comparing strings
- Specific datetime formats and values

Note DB-Library supports one language and character set at a time for error messages. For details, see “Writing internationalized DB-Library applications” on page 30.

Both Adaptive Server Enterprise and Open Server applications support localized Open Client applications. When a client connects to a server, the server determines whether or not it can support the required character set conversion (if any).

Because Open Client and Open Server support the Unicode Standard, an Open Server application can support any client, regardless of what character set it uses.

Adaptive Server Enterprise 12.5 and later support Unicode. You can use an Open Server application to perform character set conversion for earlier versions of Adaptive Server Enterprise. See “Using Open Server as a conversion gateway” on page 41.

CHAPTER 2

How Localization Works

This chapter describes how the Open Client and Open Server localization mechanism works.

This chapter covers the following topics:

Note The information in this chapter does not apply to DB-Library.

Deciding what localization values to use

Before writing an internationalized Open Client and Open Server application, you must decide how the application will localize, that is, how it will determine which language, character set, and cultural conventions to use in a given environment.

Open Client and Open Server applications can use *initial* localization values, *custom* localization values, or both:

- Initial localization values are determined at runtime, when the application allocates a context structure (cs_ctx_alloc):
  - If the LC_ALL environment variable is set, the application will use its value to localize the new context structure.
  - If the LC_ALL environment variable is not set but the LANG environment variable is set, the application will use its value to localize the new context structure.

    - If neither environment variable is set, the application uses the platform "default" entry in the locales file to localize the new context structure. The locales file, *locales.dat* is available in:
        - *$SYBASE*/*locales* directory on UNIX platforms
        - *%SYBASE%**locales* directory on Windows
- An application sets up custom localization values by calling cs_locale to fill a CS_LOCALE structure and then using the CS_LOCALE structure to change localization values for a context, connection, thread, data element, or routine.

Using initial localization values

A typical internationalized Open Client and Open Server application uses the initial localization values determined by LC_ALL, LANG, or the "default" entry in the *locales.dat* file to localize.

Initial localization values are determined at runtime, when the Open Client and Open Server application calls the CS-Library routine cs_ctx_alloc to allocate a CS_CONTEXT structure. When an application makes this call, CS-Library loads initial localization information into the new context structure.

The localization information includes:

- Language
- Character set
- Collating sequence
- Date and time formats

The loading process works as follows:

1 The application calls cs_ctx_alloc.

2 CS-Library searches the environment for the LC_ALL or LANG environment variables to determine a locale name. Table 2-1 describes this search:

Table 2-1: How CS-Library determines a locale name

Is LC_ALL defined?	Is LANG defined?	CS-Library action
Yes	N/A	Use LC_ALL's value as the locale name.
No	Yes	Use LANG's value as the locale name.
No	No	Use a locale name of "default," which means CS-Library loads one of the following: • The shipped defaults for the platform • The user-defined set assigned to the locale name "default"

3 CS-Library looks up the locale name in the *locales.dat* file to determine the associated language and character set (a collating sequence may or may not be specified). If the locale name does not exist in the *locales.dat* file, cs_ctx_alloc returns an error.

4 CS-Library loads the new context structure with the appropriate localization information.

Setting up an application to use initial localization values

If your application will use initial localization values, you should not include any special code to internationalize your application, but you do need to make sure that administrators and users know how to set environment variables for your application.

When you distribute the application, make sure that systems administrators and users understand the following:

- If LC_ALL exists, its value must correspond to the correct entry in the *locales.dat* file.
- If LANG exists, its value must correspond to the correct entry in the *locales.dat* file.
- If neither environment variable exists, the "default" entry in the *locales.dat* file must be correct (that is, it must list the language, character set, and collating sequence that the application should use).

Using custom localization values

Client-Library and Open Server applications can use custom localization values at the context, connection, thread, data element, and routine levels.

A Client-Library or Open Server application sets up custom localization values by:

1 Calling cs_locale to load a CS_LOCALE structure with specific localization values. See "The cs_locale routine" on page 11.

2 Using the loaded CS_LOCALE structure to customize a context, connection, thread, or data element. See "The CS_LOCALE structure" on page 11.

You can use command line options to run the Embedded SQL precompiler with custom localization values.

Embedded SQL applications cannot use custom values, that is, the initial localization values determined at runtime by LC_ALL, LANG, or the "default" entry in the *locales.dat* file.

Localization mechanism details

This section provides more detail about localization mechanisms. It contains information about the *locales.dat* file, localization environment variables, the CS_LOCALE structure, and the cs_locale routine.

The locales file

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

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. Open Client and Open Server applications use the *locales.dat* file to determine what localization information to load.

See Chapter 5, "Editing the Locales File."

Environment variables

On most platforms, Client-Library and Server-Library applications use the following localization environment variables:

- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGE
- LC_TIME
- LANG

Note Some systems (typically UNIX systems) automatically set localization environment variables to a specific value when a user logs in. If your system does this, either make sure that the value matches a locale name in the *locales.dat* file or reset the variables after logging in.

Table 2-2 describes how Open Client and Open Server applications use these environment variables:

Table 2-2: Environment variables related to localization

Environment variable	Set to a locale name that indicates	Used by	When
LC_ALL	Language, character set, and collating sequence to use for messages, datatype conversions, and sorting.	A Client-Library or Open Server application.	An application calls cs_ctx_alloc or cs_ctx_global. An application calls cs_locale with *type* as CS_LC_ALL and *buffer* as NULL.
		The Embedded SQL precompiler.	At application precompile time, to determine the default language and character set to use for precompiler messages.
		A precompiled Embedded SQL application.	At application runtime, when a precompiled application first calls cs_ctx_global. The precompiler generates a cs_ctx_global call for each Embedded SQL statement.

Environment variable	Set to a locale name that indicates	Used by	When
LC_COLLATE	Collating sequence (sort order) to use when sorting and comparing character data.	A Client-Library or Open Server application.	An application calls cs_locale with *type* as CS_LC_COLLATE and *buffer* as NULL.
LC_CTYPE	Character set to use for datatype conversions.	A Client-Library or Open Server application.	An application calls cs_locale with *type* as CS_LC_CTYPE and *buffer* as NULL.
LC_MESSAGE	Language and character set to use for messages.	A Client-Library or Open Server application.	An application calls cs_locale with *type* as CS_LC_MESSAGE and *buffer* as NULL.
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.	A Client-Library or Open Server application.	An application calls cs_locale with *type* as CS_LC_TIME and *buffer* as NULL.
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.	A Client-Library or Open Server application.	If an application calls cs_ctx_alloc or cs_ctx_global, Client-Library examines LANG if LC_ALL is not defined. If an application calls cs_locale, Client-Library examines LANG if cs_locale's *buffer* is NULL and the LC variable corresponding to *type* is not defined.
		The Embedded SQL precompiler.	At application precompile time, if LC_ALL is not defined.
		A precompiled Embedded SQL application.	At application runtime, if LC_ALL is not defined.

Platforms not using environment variables

This section provides information about platforms that do not use environment variables.

Desktop terminology

Some platforms use the term "environment values" instead of "environment variables." The terms mean the same thing.

The CS_LOCALE structure

The CS_LOCALE structure stores a complete set of localization information, including language, character set, collating sequence, and datetime formats.

Open Client and Open Server applications need to use a CS_LOCALE structure to define custom localization values for a context, connection, thread, data element, or routine.

❖ **To use a CS_LOCALE structure**

1 Call cs_loc_alloc to allocate a CS_LOCALE structure.

2 Call cs_locale to load the CS_LOCALE structure with the desired localization values. See "The cs_locale routine" on page 11.

3 If necessary, call cs_dt_info(CS_SET,CS_DT_CONVFMT) to change the date conversion format in the CS_LOCALE structure. See the *Open Client and Open Server Common Libraries Reference Manual*.

4 Use the loaded CS_LOCALE structure to customize a context, connection, thread, data element, or routine:

- To customize a context, call cs_config.
- To customize a connection, call ct_con_props.
- To customize a thread, call srv_thread_props.
- To define custom values for a data element, supply a pointer to the CS_LOCALE structure in a CS_DATAFMT structure.

5 To define custom values for a routine, pass a pointer to the CS_LOCALE structure to the routine.

The *cs_locale* routine

Open Client and Open Server applications use the cs_locale routine to load a CS_LOCALE structure with custom localization information.

cs_locale is declared as follows:

```
CS_RETCODE cs_locale(context, action, locale, type,
buffer, buflen, outlen)


 CS_CONTEXT *context;
 CS_INT action;
 CS_LOCALE *locale;
 CS_INT type;
```

```
CS_CHAR *buffer;
CS_INT buflen;
CS_INT *outlen;
```

When called, cs_locale performs as follows:

1 Determines what locale name to use.

   If the cs_locale *buffer* parameter is supplied, this parameter is the locale name.

   If the cs_locale *buffer* parameter is NULL, cs_locale checks for an environment variable corresponding to its *type* parameter and uses the value of this environment variable as the locale name. Make sure that the appropriate environment variables have values that correspond to entries in the *locales.dat* file.

   If an environment variable corresponding to *type* is not set, cs_locale uses a locale name of "default."

2 Looks up the locale name in the *locales.dat* file to determine the associated language, character set, and collating sequence. If cs_locale cannot find a matching entry, it returns CS_FAIL.

3 Loads the information specified by the cs_locale *type* parameter into the CS_LOCALE structure. For instance, if *type* is CS_LC_CTYPE, cs_locale loads character set information.

See the *Open Client and Open Server Common Libraries Reference Manual*.

Example: Calling *cs_locale* to Load a CS_LOCALE structure

Suppose an application is running on a machine with a *locales.dat* file containing the following entries:

```
locale = korean, korean, eucksc, korsrt
locale = C.korean, us_english, eucksc, ussrt
locale = default, us_english, iso_1, ussrt
```

where the format of an entry is:

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

Suppose further that the environment variable LC_MESSAGE has a value of "korean," and that the environment variable LC_TIME is not defined. In this environment, the application would need to make two calls to cs_locale to load a CS_LOCALE structure with the following custom values:

- "korean" as the language and "eucksc" as the character set for Client-Library and server messages

- "us_english" as the language and "eucksc" as the character set to use for conversion of datetime values

The two cs_locale calls are:

```
/*
 ** You should not specify a locale name, because
 ** cs_locale will use the value of the LC_MESSAGE
 ** environment variable as the locale name.
 */
 cs_locale(ctx, CS_SET, mylocale, CS_LC_MESSAGE,
     NULL, CS_UNUSED, NULL);

 /* Do need to specify a locale name, because
 ** there's no LC_TIME environment variable set.
 */
 cs_locale(ctx, CS_SET, mylocale, CS_LC_TIME,
     "C.korean", CS_NULLTERM, NULL);
```

After loading the CS_LOCALE structure, the application can:

- Call cs_config to copy the custom localization values into a context structure.
- Call ct_con_props to copy the custom localization values into a connection structure.
- Call srv_thread_props to copy the custom localization values into a thread structure.
- Supply the CS_LOCALE structure as a parameter to a routine that accepts custom localization values (cs_strcmp, cs_time).
- Include the CS_LOCALE structure in a CS_DATAFMT structure describing a source or destination program variable (cs_convert, ct_bind).

CHAPTER 3

Writing Internationalized Open Client and Open Server Applications

This chapter explains how to write internationalized Open Client and Open Server applications.

This chapter covers the following topics:

This chapter is not a comprehensive guide to writing Open Client and Open Server applications. Other helpful resources include:

- *Open Client and Open Server Common Libraries Reference Manual*
- *Open Client Client-Library/C Reference Manual*
- *Open Server Server-Library/C Reference Manual*
- The sample international applications, *i18n.c* for Open Client and *intlchar.c* for Open Server, shipped with Open Client and Open Server products

  See the *Open Client and Open Server Programmers Supplement* for your platform.

Writing internationalized Client-Library applications

Before writing an internationalized Client-Library application, you must decide how the application will localize, that is, how it will determine which language, character set, and cultural conventions to use in a given environment.

Client-Library applications can use initial localization values, custom localization values, or both.

Most applications use initial localization values.

For information about how initial localization values are determined and how to decide whether your application can use them, see "Deciding what localization values to use" on page 5.

Client-Library applications using initial values

If your application will use initial localization values, you should not include any special code to internationalize your application.

When you distribute your application, make sure that systems administrators know how to set environment variables. See "Setting up an application to use initial localization values" on page 7.

Client-Library applications using custom values

Client-Library applications can use custom localization values at the context, connection, and data element levels.

Open Client and Open Server applications sets up custom localization values by:

- Calling cs_locale to load a CS_LOCALE structure with specific localization values.
- Using the loaded CS_LOCALE structure to customize a context, connection, or data element.

Table 3-1 is intended to help you decide how to use custom localization values in your application:

Table 3-1: Using custom localization values in a Client-Library application

If	Then	For more information
The application needs just a single set of custom localization values (but, for whatever reason, it cannot use its initial localization values).	Customize at the context level. You can use the same CS_LOCALE structure to customize multiple contexts.	"Customizing at the context level" on page 17.
Different contexts in the application require different localization values.	Customize each context. Use different CS_LOCALE structures to customize different contexts.	"Customizing at the context level" on page 17.
Specific connections need to use localization values that differ from their parent context's localization values.	Customize those connections.	"Customizing at the connection level" on page 18.
Bind variables, conversion destination variables, or specific routines need to use custom localization values.	Customize the variables or routines.	"Customizing at the data element level" on page 20.

Customizing at the context level

You need to install custom localization values at the context level if the context's initial localization values are not acceptable.

For example, you would need to install custom localization values at the context level if different contexts in the same application required different localization values, because not all of the contexts would be created with correct initial values.

For information on how a context receives its initial localization values, see "Using initial localization values" on page 6.

Example

Suppose a Client-Library application needs to generate messages in Korean, but it is running in an environment in which the LC_ALL environment variable must be set to us_english to accommodate other applications. Because the initial us_english localization values that the context uses are not acceptable, the application needs to specify Korean localization values at the context level.

Defining custom localization values for a context

Table 3-2 describes how to define custom localization values at the context level:

Table 3-2: Installing custom values at the context level

Step	Application step	Purpose	Details
1	Call cs_loc_alloc.	Allocate a CS_LOCALE structure.	This call copies the parent context's current localization information into the CS_LOCALE structure.
2	Call cs_locale.	Overwrite the CS_LOCALE structure with custom localization values.	See "The cs_locale routine" on page 11. Open Server applications must call cs_locale with *type* as CS_LC_ALL. This ensures that Server-Library loads the CS_LOCALE structure with localization values that are internally consistent.
3	Optionally, call cs_dt_info.	Change datetime conversion formats in the CS_LOCALE structure.	See the *Open Client and Open Server Common Libraries Reference Manual*.
4	Call cs_config with *property* as CS_LOC_PROP	Customize a context.	
5	Optionally, call cs_loc_drop.	Deallocate the CS_LOCALE structure.	An application can reuse the CS_LOCALE structure before deallocating it. If necessary, the application can call cs_locale to change the localization values in the structure before reusing it.

Customizing at the connection level

A connection inherits default localization values from its parent context. You need to install custom localization values at the connection level if the connection's default localization values are not acceptable.

Example

A us_english/iso1 application that connects to a server in Spain needs to process and sort roman8 character data. Because the us_english/iso1 localization values that the connection inherits from its parent context are not acceptable, the application needs to install roman8 localization values at the connection level.

Defining custom localization values for a connection

Table 3-3 describes how to define custom localization values at the connection level.

Table 3-3: Installing custom values at the connection level

Step	Application step	Purpose	Details
1	Call cs_loc_alloc.	Allocate a CS_LOCALE structure.	This call copies the parent context's current localization information into the CS_LOCALE structure.
2	Call cs_locale.	Overwrite the CS_LOCALE structure with custom localization values.	See "The cs_locale routine" on page 11.
3	Optionally, call cs_dt_info.	Change datetime conversion formats in the CS_LOCALE structure.	See the *Open Client and Open Server Common Libraries Reference Manual*.
4	Call ct_con_props with *property* as CS_LOC_PROP.	Customize a connection.	Note that CS_LOC_PROP is a login property. An application cannot change its value after a connection is open. If an application sends a request to the server to change the language or character for the connection after the connection is open, the change will not be reflected in the value of CS_LOC_PROP. If the application calls ct_con_props to retrieve the value of CS_LOC_PROP, the retrieved locale structure will not contain the connection's current localization values.
5	Optionally, call cs_loc_drop.	Deallocate the CS_LOCALE structure.	An application can reuse the CS_LOCALE structure before deallocating it. If necessary, the application can call cs_locale to change the localization values in the structure before reusing it.

When a client application calls ct_connect to open a connection, the server determines whether it can support the requested localization. If it can, it accepts the connection as is. If it cannot, it forces the connection to an alternate language and/or character set. At this point, the client may either accept or reject the altered connection.

Customizing at the data element level

Data-element localization values can be used to customize the following:

- Bind variables (ct_bind)

  If custom localization values are not specified, bind variables use localization values from the connection with which they are associated.

- Conversion destination variables (cs_convert)

  If custom localization values are not specified, conversion destination variables use localization values from cs_convert's *context* parameter.

- cs_time and cs_strcmp behavior

  If custom localization values are not specified, these routines use the localization values associated with their *context* parameter.

You need to set up custom localization values at the data element level if the default values are not acceptable.

Example

To generate a report, an application with a us_english connection selects book titles and publication dates from a us_english database. Because the report will be sent to Paris, the publication dates must be in a standard French format.

Since the connection's us_english formats are not acceptable for the date column bind variable, the application needs to set up the bind variable to use French datetime formats.

The application can set up the bind variable for the date column to use French datetime formats as follows:

- The application loads a CS_LOCALE structure with French datetime formats.

- The application calls ct_bind to bind the date column to a character variable. In the ct_bind call, the CS_DATAFMT structure that describes the bind variable references the CS_LOCALE structure containing the French datetime formats.

When the application calls ct_fetch, the datetime value in the date column is automatically converted to a character string containing French day and month names and copied into the bound variable.

Defining custom localization values at the data element level

Table 3-4 describes how to define custom localization at the data element level.

Table 3-4: Installing custom values at the data element level

Step	Application step	Purpose	Details
1	Call cs_loc_alloc.	Allocate a CS_LOCALE structure.	This call copies the parent context's current localization information into the CS_LOCALE structure.
2	Call cs_locale.	Overwrite the CS_LOCALE structure with custom localization values.	See "The cs_locale routine" on page 11.
3	Optionally, call cs_dt_info.	Change datetime conversion formats in the CS_LOCALE structure.	See the *Open Client and Open Server Common Libraries Reference Manual*.
4	Use the CS_LOCALE structure	Customize a bind variable, destination variable, or routine.	• Customize a bind variable by using the CS_LOCALE structure in ct_bind's *datafmt* parameter. • Customize a destination variable by using the CS_LOCALE structure in cs_convert's *destfmt* parameter. • Customize cs_strcmp or cs_time's behavior by supplying the CS_LOCALE structure as a parameter to the routine.
5	Optionally call cs_loc_drop.	Deallocate the CS_LOCALE structure.	The application must not deallocate the CS_LOCALE structure until the CS_DATAFMT structure no longer references it.

Client-Library localization value precedence

Client-Library uses localization values in the following order of precedence:

1 Values defined at the data element level

2 Values defined at the connection level

3 Values defined at the context level

Client-Library localization properties

Table 3-5 lists Client-Library properties that are related to localization:

Table 3-5: Client-Library properties related to localization

Property	Description	Applies to	For more information
CS_LOC_PROP	A CS_LOCALE structure that defines localization information.	Contexts, connections	*Open Client Client-Library/C Reference Manual*
CS_CHARSETCNV	Determines whether or not the server is performing character set conversion.	Connections	*Open Client Client-Library/C Reference Manual*
CS_NOCHARSETCNV	Determines whether or not the server should perform character set conversion.	Connections	*Open Client Client-Library/C Reference Manual*

Writing internationalized Open Server applications

When writing an internationalized Open Server application, you will need to consider the following issues:

- How the application itself will localize
- How the application will support localized clients
- How the application will respond to client requests to change language and character set
- What values Server-Library localization properties should have

Localizing the application

An Open Server application's localization values determine the language in which error messages are generated and the character set and collating sequence that are used for all data operations.

Note You can use SRV_S_USESRVLANG and SRV_T_USESRVLANG properties to override the server's language when it generates error messages.

An Open Server application can use initial localization values, custom localization values, or both.

Most applications use initial localization values.

Initial localization values are determined when the application allocates its context structure. For information on how to decide whether your application can use initial localization values, see "Deciding what localization values to use" on page 5.

Open Server applications using initial values

If your application will use initial localization values, you should not include any special code to internationalize your application.

When you distribute your application, make sure that systems administrators know how to set environment variables. See "Setting up an application to use initial localization values" on page 7.

Open Server applications using custom values

If your application cannot use initial localization values, you need to install custom localization information in the application-wide context structure before calling srv_version. For information on how to do this, see Table 3-2 on page 18.

Supporting localized clients

Open Server automatically provides some support for localized clients, but your application may need to provide additional support.

Automatic support for localized clients

Open Server automatically handles some tasks associated with supporting localized clients. These tasks include:

- Performing character set conversion, if required, of both incoming and outgoing data.
- Providing Open Server error messages in the client's language and character set (provided that the SRV_T_USESRVLANG property for the client's thread structure is set to CS_FALSE).
- Providing localization information to the client in response to a client request. See "Automatic response to requests for localization information" on page 24.

For some Open Server applications, this automatic support for localized clients is sufficient, as they do not need to take any additional steps to support localized clients. However, other Open Server applications need to provide additional support for localized clients.

Automatic response to requests for localization information

After logging into an Open Server application, a client can request:

- The name of the server's character set
- The name of the server's collating sequence (sort order)
- The character set definition for the client's character set
- The sort order definition for the client's collating sequence

Clients make these requests using the sp_serverinfo system registered procedure, using Remote Procedure Call (RPC) commands.

In response, Open Server automatically returns the requested information by means of the sp_serverinfo system registered procedure. An Open Server application does not need to take any action at this point, and, in fact, is not aware that the request ever occurred.

Additional support for localized clients

An Open Server application needs to take additional steps to support localized clients under the following circumstances:

- If it passes CS-Library error messages back to clients

In this case, the Open Server application needs to ensure that CS-Library generates messages in the client's language and the Open Server application's character set. For information on how to do this, see "Localizing CS-Library messages for clients" on page 25.

- If it is acting as a gateway

  In this case, the Open Server application needs to ensure that a connection to a remote server uses the client's language and the Open Server's character set. For information on how to do this, see "Creating localized connections for Open Server gateways" on page 27.

- If a client application asks to change its language or character set

  In this case, the Open Server application needs to change the language or character set for the client thread. For information on how to do this, see "Responding to requests to change language and character set" on page 28.

Localizing CS-Library messages for clients

If an Open Server application calls a CS-Library routine with its own context structure as a parameter, any error messages that CS-Library generates as the result of the call will be in the Open Server application's language and character set.

For example, if the context parameter for a cs_convert call indicates us_english/iso_1, CS-Library generates a us_english/iso_1 message if the cs_convert call fails.

Note If a CS-Library routine takes a CS_LOCALE structure as a parameter, the localization values in this structure will override the localization values in the context parameter.

Getting CS-Library messages in the Open Server application's language and character set is acceptable only if the Open Server application logs the CS-Library messages or otherwise keeps them to itself.

However, if an Open Server application will be passing CS-Library error messages back to a client, it needs to ensure that CS-Library generates messages in the client's language and the Open Server application's character set.

The messages need to be in the client's language for the client to understand them.

The messages need to be in the Open Server application's character set for two reasons:

- Open Server applications commonly record all messages in the log file. It is important that all logged messages use the same character set.
- Open Server automatically performs character set conversion on outgoing data, including messages. Generating messages in Open Server's character set ensures that they will be correctly converted to the client's character set.

An application can ensure that messages are generated in the correct language and character set by setting up a properly localized CS_CONTEXT structure for each client thread and then using these CS_CONTEXT structures when calling CS-Library routines on behalf of clients.

Localizing a CS_CONTEXT structure for a client thread

Table 3-6 illustrates how to localize a CS_CONTEXT structure for a client thread:

Table 3-6: Localizing a CS_CONTEXT structure for a client thread

Step	Application step	Purpose	Details
1	Call cs_ctx_alloc.	Allocate a CS_CONTEXT structure for the client thread.	The context structure is allocated with initial localization values.
2	Call cs_loc_alloc.	Allocate a new CS_LOCALE structure.	This call copies the parent context's current localization information into the new CS_LOCALE structure.
3	Call srv_thread_props(GET) with *property* as SRV_T_LOCALE.	Copy the client thread's existing localization values into the new CS_LOCALE structure.	
4	Call cs_locale with *type* as CS_SYB_CHARSET.	Replace the client thread's character set information in the new CS_LOCALE structure with the Open Server application's character set information.	
5	Call cs_config with *property* as CS_LOC_PROP.	Customize the context structure.	This call copies localization information from the CS_LOCALE structure into the CS_CONTEXT structure.

Step	Application step	Purpose	Details
6	Optionally, call cs_loc_drop.	Deallocate the CS_LOCALE structure.	An application can reuse the CS_LOCALE structure before deallocating it. If necessary, the application can call cs_locale to change the localization values in the structure before reusing it.

Creating localized connections for Open Server gateways

If an Open Server application is acting as a gateway, it needs to ensure that a connection to a remote server uses the client's language and the Open Server's character set.

Note The Open Server's character set does not need to be the same as the remote server's character set, but it must be one that the remote server is capable of converting to its own.

Adaptive Server Enterprise can convert between any two Western European character sets and between any two Japanese character sets, but it cannot convert a Western European character set to a Japanese one (and vice versa). For example, Adaptive Server Enterprise can convert between ISO 8859-1 and CP850, because both of these character sets are in the Western European language group, but Adaptive Server Enterprise cannot convert between ISO 8859-1, which is Western European, and CP 1250, which is Eastern European. When Adaptive Server Enterprise is converting between character sets in different language groups, non-ASCII characters may be lost.

The simplest way for an application to do this is to set up a properly localized CS_CONTEXT structure for each client thread and then allocate remote connections for the client thread within the localized context.

See "Localizing a CS_CONTEXT structure for a client thread" on page 26.

For information on how to allocate a connection, see the *Open Client Client-Library/C Reference Manual*.

Responding to requests to change language and character set

When a client connects to an Open Server application, Open Server automatically creates a CS_LOCALE structure reflecting the client's language and character set. (The client's collating sequence is NOT included in the CS_LOCALE structure: Collating sequence information is not transmitted to the server at login time.)

For example, when a french/cp850 client logs into a us_english/iso_1 Open Server application, the Open Server application creates a french/cp850 CS_LOCALE structure. The Open Server application uses this CS_LOCALE structure to set up character set conversion routines for the client thread.

Note The information in this CS_LOCALE structure is available to Open Server programmers, who can call srv_thread_props to copy the information into a newly allocated CS_LOCALE structure.

After logging in, if a client sends a request to change its language or character set, the Open Server application must make the requested changes in the client thread's CS_LOCALE structure.

A client can request a change of language or character set in one of two ways:

- Using a language-based option command (sent with ct_command). This type of command triggers a SRV_LANGUAGE event, so the Open Server application processes the request inside a SRV_LANGUAGE event handler.
- Using an option command (sent with ct_options). This type of command triggers a SRV_OPTION event, so the Open Server application processes the request inside a SRV_OPTION event handler.

Table 3-7 describes how to change the language or character set for a client thread:

Table 3-7: Changing language or character set for a client thread

Step	Application step	Purpose	Details
1	Call cs_loc_alloc.	Allocate a CS_LOCALE structure.	This call copies the Open Server application context's current localization information into the new CS_LOCALE structure.
2	Call srv_thread_props (GET) with *property* as SRV_T_LOCALE.	Copy the client thread's existing localization values into the new CS_LOCALE structure.	

Step	Application step	Purpose	Details
3	Call cs_locale.	Overwrites the CS_LOCALE structure with the requested language or character set.	See "The cs_locale routine" on page 11.
4	Call srv_thread_props (SET) with *property* as SRV_T_LOCALE.	Set up the client thread with the new language or character set.	
5	Optionally, call cs_loc_drop.	Deallocate the CS_LOCALE structure.	An application can reuse the CS_LOCALE structure before deallocating it. If necessary, the application can call cs_locale to change the localization values in the structure before reusing it.

Note Open Server and SDK support the same character sets as Adaptive Server Enterprise.

Server-Library localization properties

Table 3-8 lists Server-Library properties that are related to localization:

Table 3-8: Server-Library properties related to localization

Property	Description	Applies to	For more information
SRV_S_USESRVLANG	Whether or not to generate messages in the server's language.	Application-wide context	*Open Server Server-Library/C Reference Manual*
SRV_T_USESRVLANG	Whether or not to generate messages in the server's language.	Thread	*Open Server Server-Library/C Reference Manual*

These properties determine whether Open Server generates error messages in the Open Server application's language or a client's language:

SRV_S_USESRVLANG is a server-wide property, set using srv_props. Its value serves as the default value for SRV_T_USESRVLANG.

SRV_T_USESRVLANG is a thread property, set using srv_thread_props. When a new thread structure is allocated, SRV_T_USESRVLANG picks up a default value from SRV_S_USESRVLANG:

- If SRV_T_USESRVLANG is CS_TRUE, Open Server generates error messages for the thread in the language of the server.
- If SRV_T_USESRVLANG is CS_FALSE, Open Server generates error messages for the thread in the language of the client.

Writing internationalized DB-Library applications

When writing a new client application, programmers should use Client-Library instead of DB-Library. The information in this section is for sites with existing DB-Library applications.

Unlike Client-Library, DB-Library does not examine environment variables to determine initial localization values. Instead, in DB-Library, initial localization values are pre-defined on a per-platform basis.

An application can change these initial values for a specific connection by changing the language name and character set name in the login record that is used to open the connection:

- To change the language name, call DBSETLNATLANG (*login*,*language_name*).
- To change the character set name, call DBSETLCHARSET (*login*,*charset_name*). An application can call DBSETLCHARSET (*login*,NULL) to specify that the server should not perform character set conversion.

An application can use a different language and character set for each server connection.

See the *Open Client DB-Library/C Reference Manual*.

Internationalizing with Embedded SQL

As an Embedded SQL application programmer, you can localize:

- The Embedded SQL precompiler
- A precompiled Embedded SQL application

Localizing the precompiler

Precompiler users can either run the precompiler with default localization values or custom localization values.

How default values are determined

If command line options are not specified, the precompiler's localization values are determined at precompiler runtime as follows:

- If the LC_ALL environment variable is set, the application uses its value to localize, matching LC_ALL's value to an entry in the locales file to determine what language and character set to use.
- If the LC_ALL environment variable is not set but the LANG environment variable is, the application uses its value to localize, matching LANG's value to an entry in the locales file to determine what language and character set to use.
- If neither environment variable is set, the application uses the "default" entry in the locales file to localize.

Specifying custom localization values

Precompiler users can use command line options to specify custom localization values for the following:

- Source file character sets

  To specify the character set of the source file that is being precompiled, use the following command line option:

  ```
  -J  locale_for_charset
  ```

  where *locale_for_charset* is a locale name that has an entry in the locales file.

  If you do not specify -J, the precompiler interprets the source file as being in the precompiler's default character set.

- Precompiler messages

To specify the language and character set that the precompiler uses for messages, use the following command line option:

```
-Z locale_for_messages
```

where *locale_for_messages* is a locale name that has an entry in the locales file.

If you do not specify -Z, the precompiler uses its default language and character set for messages.

Localizing an Embedded SQL application

An Embedded SQL application's localization values are determined at application runtime as follows:

- If the LC_ALL environment variable is set, the application uses its value to localize, matching LC_ALL's value to an entry in the locales file to determine what language and character set to use.
- If the LC_ALL environment variable is not set but the LANG environment variable is, the application uses its value to localize, matching LANG's value to an entry in the locales file to determine what language and character set to use.
- If neither environment variable is set, the application uses the "default" entry in the locales file to localize.

A typical Embedded SQL application localizes by setting the LC_ALL environment variable.

Localizing standalone utilities

Standalone utilities include isql, bcp, and defncopy. Utilities that are built on Client-Library and utilities that are built on DB-Library localize differently.

Utilities built on top of Client-Library examine environment variables to determine default localization values. See "Deciding what localization values to use" on page 5 and "Using initial localization values" on page 6.

Utilities built on top of DB-Library use platform-specific default localization values. Pre-version 11.1 and PC utilities may be built on top of DB-Library.

All utilities provide a mechanism to enable users to specify custom values for the following:

- The display character set
- The language to use for server messages
- The character set that the utility is using

See the *Open Client and Open Server Programmers Supplement* for your platform.

Tips

This section contains tips on writing and running internationalized applications.

Make sure required files are installed

Some Open Client and Open Server routines require that certain localization files be installed. If these files are not installed, Client-Library or Server-Library generates an error message in English and writes it to standard error output.

Table 3-9 lists Open Client and Open Server routines that require localization files:

Table 3-9: Open Client and Open Server routines that require localization files

Routine	Required files	File location
cs_ctx_alloc	*locales.dat*	*locales/*
	objectid.dat	*ini* (on Microsoft Windows) *config* (On UNIX)
	cslib.loc	*locales/message/language_name*
	common.loc	*locales/message/language_name*
	charset.loc	*charsets/charset_name*
	binary.srt or the sort file specified in the matching locales file entry	*charsets/charset_name*
cs_locale	*charset.loc*	*charsets/charset_name*
	binary.srt or the sort file specified in the matching locales file entry	*charsets/charset_name*
ct_init	*ctlib.loc*	*locales/message/language_name*
srv_init	*srvlib.loc*	*locales/message/language_name*

Using CS_NULLTERM with Open Client and Open Server routines

When passed to a Client-Library, Server-Library, or CS-Library routine as a buffer's length, CS_NULLTERM indicates that the value contained in the buffer is null-terminated (terminated with a single byte with value 0).

Some character sets do not support unambiguous null-terminated strings. Do not use CS_NULLTERM if your application needs to support these types of character sets.

Table 3-10 lists CS-Library, Client-Library, and Server-Library routines that allow the use of CS_NULLTERM:

Table 3-10: Open Client and Open Server routines that use CS_NULLTERM

Library	Routine	Description
CS-Library	cs_objects	Save, retrieve, or clear objects and data associated with them.
	cs_strbuild	Construct native language message strings for character sets without NULL bytes.
	cs_strcmp	Compare two strings using a specified sort order.

Library	Routine	Description
Client-Library	ct_connect	Connect to a server.
	ct_cursor	Initiate a cursor command.
	ct_debug	Manage debug library operations.
	ct_dyndesc	Perform operations on a dynamic SQL descriptor area.
	ct_labels	Define a security label or clear security labels.
	ct_options	Set or retrieve the values of server options.
	ct_remote_pwd	Define or clear passwords to be used for server-to-server connections.
Server-Library	srv_config	Set server configuration parameters.
	srv_convert	Convert data from one datatype to another.
	srv_createmsgq	Create a message queue.
	srv_createmutex	Create a mutual exclusion semaphore.
	srv_define_event	Define a user event.
	srv_deletemsgq	Delete a message queue.
	srv_deletemutex	Delete a mutex created by srv_createmutex.
	srv_describe	Describe a result row column and its data source.
	srv_envchange	Notify the client of an environment change.
	srv_getobjid	Look up the object ID for a message queue or mutex with a specified name.
	srv_getobjname	Get the name of a message queue or mutex with an identifier.
	srv_init	Initialize an Open Server.
	srv_log	Write a message to the Open Server log file.
	srv_options	Send option information to a client or receive option information from a client.
	srv_paramnumber	Return the position number of a parameter for the current remote procedure call.
	srv_regdefine	Initiate the process of registering a procedure.
	srv_regdrop	Unregister a procedure.
	srv_reginit	Begin executing a registered procedure.
	srv_regnowatch	Remove a client thread from the notification list for a registered procedure.
	srv_regparam	Describe a parameter for a registered procedure being defined, or supply data for the execution of a registered procedure.
	srv_regwatch	Add a client thread to the notification list for a specified procedure.
	srv_returnval	Define a return value for a non-remote procedure call.
	srv_sendmsg	Send a message to the client.
	srv_setustate	Set the user state field in the thread structure. The registered procedures sp_ps and sp_who display this field.
	srv_tabname	Provide the name of the table(s) associated with a set of browse mode results.

CHAPTER 4

Coded Character Set Conversion Support

This chapter explains how character set conversion works in Open Client and Open Server products.

This chapter covers the following topics:

Definitions

The following definitions apply throughout this chapter:

- A *character set* is a finite set of characters or glyphs without encoding.
- *Encoding* is the process of uniquely identifying each character in a character set with a numeric code.
- A *coded character set* is the set of numeric codes that represents a character set.

  This chapter uses the term, “coded character set,” rather than “character set,” since conversion relies on encoding.
- *Character set conversion* is the process of mapping characters in one coded character set to characters in another.

- A *direct conversion* is a conversion from one coded character set to another. Adaptive Server Enterprise and Open Server support direct conversion between character sets within the Western European and Japanese language groups.
- An *indirect conversion* is a conversion from one coded character set to another by way of an intermediate coded character set.

  Because indirect conversion allows any character set to be converted to any other character set, regardless of whether the character sets are in the same language group, it is sometimes called *universal conversion*.

Supported character sets

Note Open Server and SDK support the same character sets as Adaptive Server Enterprise.

Adaptive Server Enterprise and Open Client and Open Server products typically come with files to support the following character sets:

- Apple Macintosh Roman (mac)
- IBM Code Page 850 (cp850)
- IBM Code Page 437 (cp437)
- ISO 8859-1 (iso_1)
- ISO 8859_15 (iso_15: Latin9 - western European)
- Hewlett-Packard Roman 8 and Roman 9 (roman8 and roman9)
- Unicode UTF-8 encoding (utf8)
- Chinese following standard GB18030-2000
- Korean Code Page 949 (cp949)
- Kazakh (kz1048)

Files to support the following character sets are included with the Japanese Language Module product:

- DEC Kanji (deckanji)
- EUC JIS (eucjis)

- Shift-JIS (sjis)

For a complete list of supported languages and character sets, see the *Adaptive Server Enterprise System Administration Guide*.

Understanding coded character set conversion

Character set conversion allows clients and servers that use different coded character sets to communicate.

At the present time in Sybase systems, automatic character set conversion occurs only on the server. Adaptive Server Enterprise and Open Server support direct coded character set conversion between character sets in the Western European and Japanese language groups. These are the only direct character set conversions that Adaptive Server Enterprise and Open Server support. However, Open Server does support the conversion of any Sybase-supported character set to or from the Unicode character set in UTF-8 form. This allows Open Server to perform an indirect conversion (charset_1 to Unicode to charset_2) between any two Sybase character sets.

The Unicode standard (equivalent to ISO 10646 standard) is an international character set. Unicode has the capacity to encode virtually all characters used in the world's major written languages.

UTF-8 is a multibyte variable length encoding of Unicode that is compatible with stream-based applications. It is recommended for data exchange and storage by X/Open, POSIX, and X11 standards.

Establishing the language and character set for a connection

When a client application attempts to connect to a server, it sends a connection request specifying the following:

- Whether or not character set conversion should be disabled for the connection (through the CS_NOCHARSETCNV property for Client-Library or the DBSETLCHARSET routine for DB-Library)
- The character set to use for the connection
- The language to use for the connection

Before accepting the connection, the server checks to see if it can support the requested language and character set.

Table 4-1 summarizes Adaptive Server Enterprise and Open Server behavior at connection time:

Table 4-1: Client and server conversion behavior

Server supports client's character set	Server supports client's language	Server action	ct_connect	dbopen
Yes	Yes	Accepts the connection in the clients language and character set.	Returns CS_SUCCEED	Returns SUCCEED
No	Yes	If character set conversion is disabled, it accepts the connection but forces it to its own character set.	Returns CS_SUCCEED	Returns SUCCEED
		If character set conversion is not disabled, it rejects the connection.	Returns CS_FAIL	Returns FAIL
Yes	No	Informs the client that the connection will use: • us_english language • The client's character set	Returns CS_SUCCEED	Returns SUCCEED
No	No	If character set conversion is disabled, it accepts the connection but forces it to: • us_english language • Its own character set	Returns CS_SUCCEED	Returns SUCCEED
		If character set conversion is not disabled, it rejects the connection.	Returns CS_FAIL	Returns FAIL

Once a connection is established, the server:

- Generates all messages in the connection's negotiated language and character set
- Performs all necessary character set conversion for both incoming and outgoing data (provided that character set conversion is not disabled for the connection)

Disabling character set conversion

Client applications typically disable character set conversion for one of the following reasons:

- The client application needs to store and retrieve data in a character set that the server does not support.
- The client application will perform any necessary character set conversion.

When character set conversion is disabled, Adaptive Server Enterprise does not perform character set conversion on Transact-SQL® statements, procedure, table, view and other names, or data. The server behaves as follows:

- It assumes that Transact-SQL statements and names are in standard Transact-SQL.
- It stores data values exactly as they are sent.
- It generates messages in its default character set.

Client-Library applications can disable character set conversion for a connection by setting the CS_NOCHARSETCNV connection property to CS_TRUE before calling ct_connect to open the connection.

DB-Library applications can disable character set conversion for a connection by calling DBSETLCHARSET with *char_set* as NULL before calling dbopen to open the connection.

Using Open Server as a conversion gateway

As Open Server can convert all Sybase-supported character sets to and from Unicode (equivalent to ISO 10646 standard), UTF-8, an Open Server application can perform indirect conversions between any two Sybase-supported character sets. As a result, you can use an Open Server application to enable communication between applications and servers that use character sets in different language groups (note that loss of data may occur).

For information on how to set up an Open Server application as a conversion gateway, see “Creating localized connections for Open Server gateways” on page 27.

Files used during character set conversion

This section contains information about files used during character set conversion.

Unilib library

The Unilib® library, *libsybunic*, contains Unicode-based routines that support the conversion of any Sybase-supported character set to or from the Unicode (equivalent to ISO 10646 standard) character set in UTF-8 form.

Using custom coded character set conversion

Open Server allows applications to install custom conversion routines. Once installed, Open Server uses the custom conversion routines automatically whenever a conversion of the specified type is required.

Why install custom conversion routines?

Install custom character set conversion routines if the conversion functionality supplied with Open Server does not meet your needs. The most common reason for installing a custom conversion routine is to improve performance by replacing an indirect conversion with a direct conversion.

For example, an Open Server application could install a custom routine to convert between ISO 8859-1 and EUC JIS. This direct conversion may be faster than the indirect conversion (ISO 8859-1 to or from Unicode UTF-8 to/from EUC JIS) that is supplied with Open Server.

Writing a custom conversion routine

A custom character set conversion routine is defined as follows:

```
CS_RETCODE convfunc(context, srcfmt, srcdata,
        destfmt, destdata, destlen)
CS_CONTEXT      *context;
CS_DATAFMT      *srcfmt;
CS_VOID         *srcdata;
CS_DATAFMT      *destfmt;
CS_VOID         *destdata;
CS_INT          *destlen;
```

where:

- *context* is a pointer to a CS_CONTEXT structure.

- *srcfmt* is a pointer to a CS_DATAFMT structure describing the source data. *srcfmt→maxlength* describes the actual length, in bytes, of the source data.
- *srcdata* is a pointer to the source data.
- *destfmt* is a pointer to a CS_DATAFMT structure describing the destination data. *destfmt→maxlength* describes the actual length, in bytes, of the destination data space.
- *destdata* is a pointer to the destination data space.
- *destlen* is a pointer to an integer. If the conversion is successful, the custom routine should set **destlen* to the number of bytes placed in **destdata*.

cs_config is the only CS-Library, Client-Library, or Server-Library routine that can be called from within a custom conversion routine.

CS-Library raises a CS-Library error if the custom routine returns any value other than CS_SUCCEED. The type of error that CS-Library raises depends on the value that the custom routine returns.

Table 4-2 lists the legal return values for a custom conversion routine:

Table 4-2: Return values for a custom conversion routine

Return value	Indicates
CS_SUCCEED	The conversion is successful.
CS_TRUNCATED	The conversion resulted in truncation.
CS_MEM_ERROR	A memory allocation failure has occurred.
CS_EBADXLT	Some characters could not be converted.
CS_ENOXLT	The requested conversion is not supported.
CS_EDOMAIN	The source value is outside the domain of legal values for the datatype.
CS_EDIVZERO	Division by zero is not allowed.
CS_EOVERFLOW	The conversion resulted in overflow.
CS_EUNDERFLOW	The conversion resulted in underflow.
CS_EPRECISION	The conversion resulted in loss of precision.
CS_ESCALE	An illegal scale value was encountered.
CS_ESYNTAX	The conversion resulted in a value that is not syntactically correct for the destination type.
CS_ESTYLE	The conversion operation was stopped due to a style error.

Installing a custom conversion routine

An application calls cs_manage_convert to install a custom conversion routine. For information on cs_manage_convert, see the *Open Client and Open Server Common Libraries Reference Manual*.

Character set conversion in Adaptive Server Enterprise releases prior to 4.9

In releases earlier than 4.9, Adaptive Server Enterprise data servers do not perform character set conversion. If your client application communicates with a pre-release 4.9 Adaptive Server Enterprise but uses a different character set from the server, international characters may not be represented correctly.

To solve the problem, you can:

- Change your client application's character set to match that of the Adaptive Server Enterprise, or

- Install custom character set conversion routines using cs_manage_convert and call cs_convert to convert the data before sending it to the server.

Mainframe support

Mainframe systems commonly use run-encoded character encoding, which provides escapes into other character encoding within a single character string.

Open Client and Open Server products do not support this mechanism.

CHAPTER 5

Editing the Locales File

This chapter describes the locales file and explains how to change it.

This chapter covers the following topics:

The locales file is named *locales.dat* and resides in the *locales* subdirectory of the Sybase directory tree. See Appendix A, “Directories and Files Related to Internationalization.”

Quick start

This section summarizes the process of adding or changing a locale definition. For more detailed information on the locales file and how to edit it, read the remainder of the chapter.

❖ **To add or change a locale definition**

1. Make a copy of the locales file (*locales.dat*), found in the *locales* directory, in case problems occur with the edited version.
2. Edit the locales file: Add or change the desired entries in the appropriate platform-specific section.
3. Update localization environment variables (LC_ALL, LC_CTYPE, LC_MESSAGE, LC_TIME, LANG) as appropriate.
4. 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.

When to edit the locales file

If the predefined locales file entries do not meet your needs, you can either change them or add entries that define new locale names. For example, you may want to edit the locales file to do the following:

- Change the language, character set, or collating sequence specified in a locale entry.
- Add locale definitions, such as those needed for new language modules.
- Match locale names used by non-Sybase software. For example, one Sybase predefined locale name is “fr”:

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

  If a non-Sybase application requires the LC_ALL environment variable to have a value of “french” and you want your Open Client and Open Server applications to use LC_ALL to localize with this locales file entry, you need to add a new entry or change the locale name specified in the existing entry as follows:

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

Locales file sections and entries

The locales file resides in the Sybase release directory under the *locales* subdirectory.

The locales file contains:

- Standard sections (see Table B-2 on page 81)
- Platform-specific sections containing locale definition entries

Locale definition entries

The locales file has 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. *locale_name* is usually vendor-specified, based on POSIX terminology. Comments at the end of the locales file 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. *sortorder_name* is optional. If not specified, Open Client and Open Server products use a binary collating sequence.

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

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

Locales file example

The following fragment illustrates some platform-specific sections in a locales file:

```
[aix]
      locale = en_US, us_english, iso_1
      locale = en_US.ISO8859-1, us_english, iso_1
      locale = en_JP, us_english, eucjis
      locale = FR_FR.IBM-850, french, cp850
      locale = fr_FR.ISO8859-1, french, iso_1
      locale = fr_CA, french, iso_1
      locale = Fr_CA.IBM-850, french, cp850
      locale = fr_CA.ISO8859-1, french, iso_1

[linux]
      locale = GERMAN, german, iso_1
      locale = de, german, iso_1
      locale = de_AT, german, iso_1
      locale = de_AT.437, german, cp437
      locale = de_AT.850, german, cp850
      locale = CHINESE, chinese, eucgb
      locale = zh_CN, chinese, eucgb
      locale = zh_CN.GB18030, chinese, gb18030
```

```
locale = zh_CN.gbk, chinese, eucgb
locale = zh_TW, tchinese, big5
```

Editing the locales file

Before editing the locales file:

- Review the entries listed for your platform to see if a suitable entry already exists. If so, you do not have to edit the locales file.
- Make a backup copy of the original locales file, in case problems occur with the edited version.

Adding or changing entries

To add a new entry to the locales file or to change an existing entry:

1 Choose a value for *locale_name.*

locale_name can have any value. Sybase recommends names of the form *language.territory*.

2 Determine the value to use 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 to use 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 to use 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's names (without the .*srt*).

5 In the appropriate platform-specific section of the locales file, type in or change the appropriate entry.

After you make the change:

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

Deleting entries

It is not necessary to delete entries from the locales file, even if applications no longer use them. If you decide to delete an entry, make sure no application uses it.

CHAPTER 6

Creating or Changing Collating Sequences

This chapter explains how to create and change collating sequence (sort order) files.

This chapter covers the following topics:

Quick start

This section summarizes the process of creating and changing sort order files. For more detailed information, read the remainder of the chapter.

❖ **To create or change a sort order file**

1 Copy one of the shipped **.srt* files and rename it, keeping the *.srt* suffix.

Note Do not modify the **.srt* files shipped with the product. Instead, make copy of the original **.srt* file and then modify its copy.

---

2 Edit the newly created file, changing or adding entries as follows:

- Specify general entries for the [sortorder] section, including "class," "id," "menuname," "charset," "preference," and "description."

- List ligatures, using the entry form “lig = *value*.” Group ligature entries before character entries.
- List all the character set’s characters and glyphs in the desired primary sort order, using the entry form “char = *value*.”
- For the secondary sort order, add values horizontally to the primary sort order entries, using the entry form “char = *value1, value2, ...*”
- For sorting that is not case sensitive, put equal signs between uppercase and lowercase counterparts.

3 Save the new *.srt* file in the *charsets* directory under the *charset_name* subdirectory.

4 Edit locales file entries, as appropriate, to refer to the new collating sequence file.

About collating sequences

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

Collating sequence definitions are built on top of character set definitions, but languages that use the same character set can order characters differently. For example, in Spanish “Co” comes before “Cho,” because “Ch” is considered to be a single letter; in English “Cho” alphabetically precedes “Co.”

Ordering conventions can also vary between languages for letter-diacritic combinations. For instance, “Å” might come after “z,” even though “a” (without diacritics) comes before “b.”

This section discusses some common considerations in defining collating sequences, but it is not intended to be comprehensive. Please refer to general references on collating sequences.

Definitions

If you are unfamiliar with Sybase collating sequences, the following definitions may be useful:

- The collating sequence’s *primary sort order* is the vertical sequence of lines beginning with “char=”.

- A primary entry's *secondary sort order* is the horizontal sequence of characters on a single "char =" line.

Types of sorts

There are many ways to sort characters. Open Client and Open Server collating sequence file can use one or more of the types of sorts listed in Table 6-1:

Table 6-1: Types of sort orders

Type of sort	Description
Single-level	Characters sort according to their primary sort order value. A character that appears on a line higher in the vertical list of "char=" entries always sorts before a character that appears on a line lower in the list.
Two-level	Characters sort according to their primary and secondary sort order values. If all the characters in two strings have the same primary sort values, then the characters' secondary sort values are used to break the ordering tie. If two characters appear on the same "char =" line, the one furthest to the left sorts first. For example, suppose a sort order file contains: char = A,a,Ä,ä char = B,b char = C,c,Ç,ç Some strings using these characters would sort as follows: ABC ÄBC äbc acb ächb Because the strings ABC, ÄBC, and äbc have the same primary values, they are ordered by their secondary sort values. acb and ächb are similarly sorted according to secondary values. äbc is ranked before acb because b has an earlier primary value than c.
One-to-two	A single character that is sorted as multiple characters is called a *ligature*. For example, the German character "ß" is sorted as "ss."
Two-to-one	A 2-character string that is sorted as 1 character is called a *sort double*. For example, the Spanish character string "ch" is sorted as one character that comes between "c" and "d".

Determining case sensitivity

Most collating sequence files list all variants of a single letter on one char = line.

A collating sequence that is case sensitive lists uppercase and lowercase variants of a letter in the order in which they are to be sorted and separates them with a comma. For example:

```
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2,0xC3,0xE3
;A, a, A-grave, a-grave, A-acute, a-acute, A-tilde, a-tilde,
;A-diaeresis, a-diaeresis
;
char = 0x42,0x62
;letter B, b
```

A collating sequence that is not case sensitive lists the uppercase and lowercase variants of a letter in any order and joins them with an equals sign. For example:

```
char = 0x41=0x61,0xC0=0xE0,0xC1=0xE1,0xC2=0xE2,0xC3=0xE3
;A, a, A-grave, a-grave, A-acute, a-acute, A-tilde, a-tilde,
;A-diaeresis, a-diaeresis
;
char = 0x42=0x62
;letter B, b
```

When to create a custom collating sequence file

On most platforms, Open Client and Open Server products include the standard collating sequence files described in Table 6-2:

Table 6-2: Commonly-supplied collating sequences

File name	Description
binary.srt	Ordering corresponds to the internal binary value for each character. *binary.srt* contains the entry "binary = true". No localization file is necessary for this sort order.
dictionary.srt	Dictionary order, case sensitive. Primary lexicographic ordering with uppercase letters before their lowercase counterparts. Secondary ordering for accented characters. The file name varies according to language. For example, the Spanish version is called *espdict.srt*.
noaccents.srt	Dictionary order, accent insensitive, not case sensitive. Intermingles words that begin with an unaccented letter and words that begin with the letter's accented counterparts. The file name varies according to language. For example, the Spanish version is called *espnoac.srt*.
nocase.srt	Dictionary order, not case sensitive. Intermingles words that begin with an uppercase letter with words that begin with the lowercase counterpart. The file name varies according to language. For example, the Spanish version is called *espnocs.srt*.
nocasepref.srt	Dictionary order, not case sensitive, with preference for uppercase only when there is a lowercase equivalent.

If a language you are using has further collating sequence requirements, you can create a custom collating sequence file according to the guidelines in "About collating sequence files" on page 58.

About collating sequence files

Sybase collating sequence files are named **.srt* and are located in the *charsets/charset_name/* directory. All collating sequence files use standard Sybase external localization file syntax.

See Appendix B, "External Localization File Syntax."

Collating sequence file sections and entries

All collating sequence files include the following elements:

- The comment line, copyright section, and file format section, described in Table B-2 on page 81.
- General entries, described in Table 6-4 on page 62.
- Ligature entries, described in step 3 under “Creating a custom collating sequence file” on page 61.
- Character entries, described in steps 4, 5, 6, and 7 under “Creating a custom collating sequence file” on page 61.

Writing characters in a collating sequence file

There are three ways to write characters in a collating sequence file entry:

- By typing the hexadecimal character encoding for the character. For example:

```
char = 0x20 ;( ) space
char = 0x3D ;(=) equals sign
```

- By typing the character, quoted. For example:

```
char = " " ;( ) space
char = "=" ;(=) equals sign
```

- By typing the character itself. For example:

```
char = A, a
char = B, b
```

Table 6-3 classifies characters according to how they can be written in collating sequence file entries:

Table 6-3: Writing characters in a collating sequence file entry

Type of character	Can be written as hexadecimal numbers?	Can be typed in with quotes?	Can be typed in without quotes?
Non-printable characters and characters that do not appear on the keyboard	Yes	No	No
Space (“ ”) Equals sign (“=”) Comment character Escape character List separator character	Yes	Yes	No
Backslash ("\")	Yes	Yes, but must be doubled inside of quotes ("\\")	No
All other characters	Yes	Yes	Yes

The *preference* keyword and the *order by* clause

A collating sequence file that is not case sensitive can use a preference entry to indicate whether letters to the left of the equal sign should sort before letters to the right of the equal sign when sorting output generated as the result of a select statement with an order by clause.

For example, suppose that a collating sequence file contains the following entries:

```
char = A=a, Á=á
char = B=b
```

If preference=true, then order by output will sort as follows:

```
Aab
aAb
Aáb
```

If preference=false, then order by output can sort either as:

```
aAb
Aab
Aáb
```

or

```
Aab
aAb
Aáb
```

The preference keyword:

- Applies only to sort orders that are not case sensitive
- Affects only sorts that occur as the result of an order by clause

If preference=true, then characters to the left of the equal sign sort first. If preference=false, then characters to the left of the equal sign may not sort first.

The preference keyword has a default value of "true." That is, if a collating file does not contain a preference entry, order by sorts give precedence to characters to the left of the equal sign.

Most typically, preference=true means that uppercase characters sort before lowercase characters.

Creating a custom collating sequence file

This section explains how to create a custom collating sequence file. Before you begin, please read this entire section and familiarize yourself with the collating sequence files included with your Open Client and Open Server products.

"Collating sequence file example" on page 65 illustrates a collating sequence file.

Appendix B, "External Localization File Syntax" provides general information about localization file syntax.

To create or change a collating sequence file:

1 If you plan to use a shipped *.srt* file as a model, be sure to copy and rename it so you do not overwrite the original file. The new file's name must include the *.srt* suffix. In addition, a descriptive name helps to associate the file with the language it supports.

2 Determine the values for general entries. Table 6-4 describes these entries:

Table 6-4: .srt file general entries

Entry keyword	Description	Required	Entry value
class	The sort order class. Currently, class 1 for 8-bit character sets is the only supported class.	Yes	0x01d
id	A unique hexadecimal number that identifies the collating sequence.	Yes	For user-defined collating sequences, ID must have a value of 0xC9 through 0xFF. Sybase reserves hexadecimal 0x00 through 0xC8.
menuname	The name of the collating sequence as it is to appear in the sybinit program.	Yes	A string no longer than 64 characters is recommended. sybinit truncates strings to 64 characters. This value is user-defined.
name	The name of the collating sequence.	No	A string no longer than 30 characters. This value is user-defined.
charset	The character set with which this collating sequence file is intended for use. This is also the name of the directory in which this collating sequence file will reside.	Yes	The value must match a character set subdirectory name in the Sybase directory tree.
preference	For sort orders that are not case sensitive, whether to give preference to characters to the left of the equals sign when sorting output generated by a select statement with an order by clause.	No	False – no preference. True – preference for characters to the left of the equals sign. A value of "true" has a greater performance impact than "false." The default is "true."
description	Phrase that describes the collating sequence. Stored with the collating sequence.	No	A string no longer than 255 characters. This value is user-defined.

3 Determine whether there are any ligatures. A *ligature* is a single character that is sorted as multiple characters. If there are ligatures:

- Place the ligature ("lig") entries together, preceding the "char" entries.
- Include both the uppercase and lowercase forms of a ligature, if applicable.

The syntax for a case-sensitive ligature is:

```
lig = value, after characters ;case-sensitive sort
```

where:

- *characters* is a string representing the characters after which the ligature will sort.
- *value* is the hexadecimal encoding for the ligature character, or the typed or quoted ligature character.

The syntax for a ligature that is not case sensitive:

```
lig = value1=value2, after characters ;case-
insensitive sort
```

where:

- *value1* and *value2* are the hexadecimal encodings for the uppercase and lowercase ligature characters, or the typed or quoted ligature characters.
- *characters* is a string representing the characters after which the ligature will sort.

The following example shows ligature entries in a collating sequence file that is not case sensitive for ISO 8859-1:

```
lig = 0xC6, after AE ;diphthong AE, A with E
lig = 0xE6, after ae ;diphthong ae, a with e
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2x
;varieties of letter A
char = 0x42,0x62 ;B, b
. . .
```

4 Vertically list all the character entries for the sort order. This vertical list is the primary sort order.

The syntax for a character entry is:

```
char = value
```

where *value* is the hexadecimal code set encoding for the character, or the typed or quoted character.

For example:

```
char = 0x41   ;ISO 8859-1 code set.
```

5 If applicable, add secondary sort order information to the file as follows:

- For a case-sensitive sort order, put the lowercase variant to the right of the uppercase character (if you want the uppercase character to take precedence). Separate the characters with the list separator character.
- For a sort order that is not case sensitive, put equal signs between each uppercase character and its lowercase equivalent (including accented characters).
- Put a character and its variants in relative order to each other. For example, the French "é" goes to the right of "e." Make sure these characters are not ligatures or separate primary sort order entries. Separate variants with the list separator character.

The following example shows secondary sort order information for a Latin alphabet, case-sensitive sort order:

```
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2,
0xC3,0xE3,0xC4,0xE4,0xC5,0xE5
;A, a, A-grave, a-grave, A-acute, a-acute,
;A-circumflex, a-circumflex, A-tilde, a-tilde,
;A-diaeresis, a-diaeresis, A-ring, a-ring
. . .
char = 0x4E,0x6E,0xD1,0xF1 ;N, n, N-tilde, n-tilde
. . .
```

6 Determine whether there are any sort doubles. A *sort double* or *digraph* is a pair of characters that is sorted as a single character. If there are any sort doubles:

- List each sort double as a separate "char" entry.
- For case-sensitive sorting, put all permutations of the sort double in the desired sort order.

The syntax for a sort double is:

```
char = value1value2
```

where:

- *value1* is the first character in the sort double pair,

- *value2* is the second character in the pair.

If *value1* and *value2* are written as hexadecimal numbers, use a leading '0x' with *value1* but not with *value2*. For example:

```
char = 0x4348,0x4368,0x6348,0x6368 ;CH,Ch,cH,ch
```

value1 and *value2* can also be typed or quoted characters. For example:

```
char = CH, Ch, cH, ch
```

or

```
char = "CH", "Ch", "cH", "ch"
```

The following example shows the placement of the Spanish sort double "ch" in a case-sensitive *.srt* file for the iso_1 (ISO 8859-1) character set:

```
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2
;varieties of letter A
char = 0x42,0x62 ;B, b
char = 0x44,0x64,0xC7,0xE7 ;C, c, C-cedilla, c-
cedilla
char = 0x4348,0x4368,0x6348,0x6368 ;CH,Ch,cH,ch
. . .
```

7 Include all other characters in the vertical list, such as non-printable characters, characters not on a keyboard, symbols, and characters related to linguistic style. Use "char" or "lig" entries, as appropriate. Be sure to group all "lig" entries together before "char" entries.

For information on how to write nonalphabetic characters in a collating sequence file, see Table 6-3 on page 60.

8 Save the new *.srt* file in the *charsets* directory under the *charset_name* subdirectory.

9 Edit locales file entries, as appropriate, to refer to the new collating sequence file. See Chapter 5, "Editing the Locales File."

Collating sequence file example

This section contains an example of a case-sensitive collating sequence file.

Actual collating sequence files are included in your Sybase directory tree as *charsets/charset_name/*.srt*.

```
; semi-colon is the comment character
 [sortorder]
 ;=========================================================================
 ;
 ; @(#)dictionary.srt
 ;
 ; Sort Order Overview:
 ; --------------------
 ; Based on the ISO 8859-1 ("Latin 1") character set, this sort order is
 ; a case-sensitive ordering. Upper case letters always sort before their
 ; lower case counterparts.
 ;
 ; It is useful for at least the English, French and German languages,
 ; and may work for many others.
 ;
; Ligatures, Sort-Doubles, etc.:
 ; ------------------------------
 ; AE, ae ligatures
 ; German sharp-s ligature with "ss"
 ;
 ; The ordering:
 ; --------------
 ; first all non-alphanumeric characters in binary order
 ; followed by all numeric digits
 ; then all alphabetic characters used in English, French and German
 ; and ended by all alphabetic characters not used in English, French
 ; or German
 ;=========================================================================
 class = 0x01 ; Class `1' sort order
 id = 0x33 ; Unique ID # (51) for the sort order
 name = dictionary_iso_1
 menuname = "General purpose dictionary ordering."
 charset = iso_1
 description = "General purpose dictionary sort order for use with several
 Western-European languages including English, French, and German.  Uses the
 ISO 8859-1 character set and is case-sensitive."
 ;
 ; ligatures for English, French, and German
 lig = 0xC6, after AE  ;AE ligature
 lig = 0xE6, after ae ;ae ligature
 lig = 0xDF, after ss ;small german letter sharp s
 ;
 ; Control characters
 char = 0x01 ;(SOH) start of heading
 char = 0x02 ;(STX) start of text
 char = 0x03 ;(ETX) end of text
```

```
char = 0x04 ;(EOT) end of transmission
char = 0x05 ;(ENQ) enquiry
char = 0x06 ;(ACK) acknowledge
char = 0x07 ;(BEL) bell
char = 0x08 ;(BS) backspace
char = 0x09 ;(HT) horizontal tab
char = 0x0A ;(LF) newline, or line feed
char = 0x0B ;(VT) vertical tab
char = 0x0C ;(FF) form feed
char = 0x0D ;(CR) carriage return
char = 0x0E ;(SO) shift out
char = 0x0F ;(SI) shift in
char = 0x10 ;(DLE) data link escape
char = 0x11 ;(DC1) device control 1
char = 0x12 ;(DC2) device control 2
char = 0x13 ;(DC3) device control 3
char = 0x14 ;(DC4) device control 4
char = 0x15 ;(NAK) negative acknowledge
char = 0x16 ;(SYN) synchronous idle
char = 0x17 ;(ETB) end transmission blk
char = 0x18 ;(CAN) cancel
char = 0x19 ;(EM) end of medium
char = 0x1A ;(SUB) substitute
char = 0x1B ;(ESC) escape
char = 0x1C ;(FS) file separator
char = 0x1D ;(GS) group separator
char = 0x1E ;(RS) record separator
char = 0x1F ;(US) unit separator
;
; All non-alphanumeric characters, including puntuation.
; These are sorted by their numerical ordering, based on the
; ISO 8859-1 standard, for clarity and consistency.
;
char = 0x20 ;( ) space
char = 0x21 ;(!) exclamation mark
char = 0x22 ;(") quotation mark
char = 0x23 ;(#) number sign
char = 0x24 ;($) dollar sign
char = 0x25 ;(%) percent sign
char = 0x26 ;(&) ampersand
char = 0x27 ;(') apostrophe
char = 0x28 ;(() left parenthesis
char = 0x29 ;()) right parenthesis
char = 0x2A ;(*) asterisk
char = 0x2B ;(+) plus sign
char = 0x2C ;(,) comma
```

```
char = 0x2D ;(-) hyphen, minus sign
char = 0x2E ;(.) full stop
char = 0x2F ;(/) solidus
char = 0x3A ;(:) colon
char = 0x3B ;(;) semicolon
char = 0x3C ;(<) less-than sign
char = 0x3D ;(=) equals sign
char = 0x3E ;(>) greater-than sign
char = 0x3F ;(?) question mark
char = 0x40 ;(@) commercial at
char = 0x5B ;([) left square bracket
char = 0x5C ;(\) reverse solidus
char = 0x5D ;(]) right square bracket
char = 0x5E ;(^) circumflex accent
char = 0x5F ;(_) low line
char = 0x60 ;(`) grave accent
char = 0x7B ;({) left curly bracket
char = 0x7C ;(|) vertical line
char = 0x7D ;(}) right curly bracket
char = 0x7E ;(~) tilde
char = 0x7F ;delete, or rubout
char = 0x80 ; undefined
char = 0x81 ; undefined
char = 0x82 ; undefined
char = 0x83 ; undefined
char = 0x84 ; undefined
char = 0x85 ; undefined
char = 0x86 ; undefined
char = 0x87 ; undefined
char = 0x88 ; undefined
char = 0x89 ; undefined
char = 0x8A ; undefined
char = 0x8B ; undefined
char = 0x8C ; undefined
char = 0x8D ; undefined
char = 0x8E ; undefined
char = 0x8F ; undefined
char = 0x90 ; undefined
char = 0x91 ; undefined
char = 0x92 ; undefined
char = 0x93 ; undefined
char = 0x94 ; undefined
char = 0x95 ; undefined
char = 0x96 ; undefined
char = 0x97 ; undefined
char = 0x98 ; undefined
```

```
char = 0x99 ; undefined
char = 0x9A ; undefined
char = 0x9B ; undefined
char = 0x9C ; undefined
char = 0x9D ; undefined
char = 0x9E ; undefined
char = 0x9F ; undefined
char = 0xA0 ;no-break space
char = 0xA1 ;inverted exclamation mark
char = 0xA2 ;cent sign
char = 0xA3 ;pound sign
char = 0xA4 ;currency sign
char = 0xA5 ;yen sign
char = 0xA6 ;broken bar
char = 0xA7 ;paragraph sign, section sign
char = 0xA8 ;diaeresis
char = 0xA9 ;copyright sign
char = 0xAA ;feminine ordinal indicator
char = 0xAB ;left angle quotation mark
char = 0xAC ;not sign
char = 0xAD ;soft hyphen
char = 0xAE ;registered trade mark sign
char = 0xAF ;macron
char = 0xB0 ;ring above or degree sign
char = 0xB1 ;plus/minus (+/-) sign
char = 0xB2 ;superscript 2
char = 0xB3 ;superscript 3
char = 0xB4 ;acute accent
char = 0xB5 ;micro sign
char = 0xB6 ;pilcrow or paragraph sign
char = 0xB7 ;middle dot
char = 0xB8 ;cedilla
char = 0xB9 ;superscript 1
char = 0xBA ;masculine ordinal indicator
char = 0xBB ;right angle quotation mark
char = 0xBC ;vulgar fraction one quarter
char = 0xBD ;vulgar fraction one half
char = 0xBE ;vulgar fraction three quarter
char = 0xBF ;inverted question mark
char = 0xD7 ;multiplication sign
char = 0xF7 ;division sign
;
; Digits
char = 0x30 ;(0) digit zero
char = 0x31 ;(1) digit one
char = 0x32 ;(2) digit two
```

```
char = 0x33 ;(3) digit three
char = 0x34 ;(4) digit four
char = 0x35 ;(5) digit five
char = 0x36 ;(6) digit six
char = 0x37 ;(7) digit seven
char = 0x38 ;(8) digit eight
char = 0x39 ;(9) digit nine
;
; Latin Alphabet
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2,0xC3,0xE3,0xC4,0xE4,0xC5,0xE5
;     A, a, A-grave, a-grave, A-acute, a-acute, A-circumflex,
;     a-circumflex, A-tilde, a-tilde, ;A-diaeresis, a-diaeresis,
;     A-ring, a-ring
char = 0x42, 0x62 ;letter B, b
char = 0x43, 0x63, 0xC7, 0xE7
;     letters C, c, C-cedilla, c-cedilla
char = 0x44, 0x64 ;letter D, d
char = 0x45, 0x65, 0xC8, 0xE8, 0xC9, 0xE9, 0xCA, 0xEA, 0xCB, 0xEB
;     E, e, E-grave, e-grave, E-acute, e-acute, E-circumflex,
;     e-circumflex, E-diaeresis, e-diaeresis
char = 0x46, 0x66 ;letter F, f
char = 0x47, 0x67 ;letter G, g
char = 0x48, 0x68 ;letter H, h
char = 0x49, 0x69, 0xCC, 0xEC, 0xCD, 0xED, 0xCE, 0xEE, 0xCF, 0xEF
;     I, i, I-grave, i-grave, I-acute, i-acute, I-circumflex,
;     i-circumflex, I-diaeresis, i-diaeresis
char = 0x4A, 0x6A ;letter J, j
char = 0x4B, 0x6B ;letter K, k
char = 0x4C, 0x6C ;letter L, l
char = 0x4D, 0x6D ;letter M, m
char = 0x4E, 0x6E, 0xD1, 0xF1
     ;letters N, n, N-tilde, n-tilde
char = 0x4F,0x6F,0xD2,0xF2,0xD3,0xF3,0xD4,0xF4,0xD5,0xF5,0xD6,0xF6,0xD8,0xF8
;     O, o, O-grave, o-grave, O-acute, o-acute, O-circumflex,
;     o-circumflex,O-tilde, o-tilde, O-diaeresis, o-diaeresis,
;     O-stroke, o-stroke
char = 0x50, 0x70 ;letter P, p
char = 0x51, 0x71 ;letter Q, q
char = 0x52, 0x72 ;letter R, r
char = 0x53, 0x73 ;letter S, s
char = 0x54, 0x74 ;letter T, t
char = 0x55,0x75,0xD9,0xF9,0xDA,0xFA,0xDB,0xFB,0xDC,0xFC
;     U, u, U-grave, u-grave, U-acute, u-acute,
;     U-circumflex, u-circumflex, U-diaeresis, u-diaeresis
char = 0x56, 0x76 ;letter V, v
char = 0x57, 0x77 ;letter W, w
```

```
char = 0x58, 0x78 ;letter X, x
char = 0x59, 0x79, 0xDD, 0xFD, 0xFF
;     letters Y, y, Y-acute, y-acute, y-diaeresis
char = 0x5A, 0x7A ;letter Z, z
;
; Alpha characters not used in English, French or German:
char = 0xD0, 0xF0 ;icelandic capital letter Eth, small letter eth
char = 0xDE, 0xFE ;icelandic capital letter Thorn, small letter thorn
```

APPENDIX A

Directories and Files Related to Internationalization

This appendix describes the Open Client and Open Server directories and files that are related to internationalization and localization.

This chapter covers the following topics:

Overview

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

- *locales* directory, which contains files that your application uses to load localization information. It also contains language-specific message files.
- *charsets* directory, which contains conversion and collating sequence files for each supported character set.
- *config* directory on UNIX and *ini* directory on Microsoft Windows, which contains the global object identifiers file.
- *collate* directory, which the Adaptive Server Enterprise uses for sorting. Each character set comes with one or more sort orders that Adaptive Server Enterprise uses to collate data.

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

Note The installation process automatically loads any additional Open Client and Open Server Language Module for connectivity into the Sybase release directory in the correct locations.

The *locales* directory

The *locales* directory contains:

- The locales file (*locales.dat*), which maps locale names to languages, character sets, and collating sequences.
- The *message* directory, which contains localized error messages for Open Client and Open Server products, 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 locales file

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

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

See Chapter 5, “Editing the Locales File.”

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.

language_name subdirectories

If you 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

Unicode directory

All Open Client and Open Server message files use the Unicode UTF-8 character set, converting messages from UTF-8 to other character sets as needed.

The *charsets* directory

The *charsets* directory contains:

- A *charset_name* subdirectory for each character set. Each *charset_name* subdirectory contains collating sequence files for each supported character set.

- *unicode* directory, which contains Unicode conversion files used by Unilib.

Collating sequence 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*

If these files do not meet your needs, you can create a custom collating sequence file. For information on how to do this, see "Creating a custom collating sequence file" on page 61.

Collating sequences are specified in locales file entries. If a locales file entry does not specify a collating sequence, then a binary sort order is used with the locale. See Chapter 6, "Creating or Changing Collating Sequences."

Unicode conversion files

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

The *config* and *ini* directories

The *config* directory (on UNIX) and the *ini* directory (on Microsoft Windows) contain the global identifiers file (*objectid.dat*).

The global object identifiers file

The global object identifiers file, *objectid.dat*, associates a unique global object identifier with all local names that might be used for the 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.

Object identifiers file sections and entries

The *objectid.dat* file contains a section for each class of object.

Object class entries have the following 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.

Object identifiers file 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]
     1.3.6.1.4.1.897.4.6.6 = csfkrb5
```

Editing object identifiers file

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

APPENDIX B

External Localization File Syntax

This appendix describes external localization file syntax and shows a sample file. Use this information when creating or updating external localization files, such as the locales file (locales.dat) and collating sequence files (*sort_order_name.srt*).

This chapter covers the following topics:

Localization file syntax rules

All external localization files observe the following basic syntax rules:

- *Comments* start with a comment character and continue to the end of the line. The first character in the first line of the file is defined to be the comment character for the file.
- *Sections* begin with a section heading and contain entries. Section headings use left and right delimiters. A section heading's maximum length is 63 bytes, including delimiters.

  The first line in the file that does not begin with a comment character defines section heading delimiters for the file. Its first character is defined to be left delimiter and its last character is defined to be the right delimiter.
- *Entries* take the following form:

  keyword = value_list

  where:

- *keyword* is the entry keyword and can be up to 63 bytes long.
- *value_list* is a list of one or more values separated by the list separator character. Each value can be a quoted or unquoted string or a hexadecimal number. If no *value_list* is present, the entry keyword is assigned a single zero-length string (that is, a string that contains only a NULL terminator) as its value.

  value_list can span multiple lines if each line except the last ends with the escape character.

  value_list can be up to 511 bytes long.

Only one entry can appear on a line. An entry can be preceded by tabs and spaces.

- *Values* can be hexadecimal numbers or quoted or unquoted strings.
  - Unquoted strings beginning with "0x" are interpreted as hexadecimal numbers.
  - Strings do not require quotes unless they contain list separators or spaces. List separators and spaces that occur inside a quoted string are treated as though they were preceded by the escape character.
  - You can use either apostrophes or quotation marks to quote strings. Apostrophes (') can appear in strings delimited by quotation marks ("string") and quotation marks can appear in strings delimited by apostrophes.

    If either the apostrophe or quotation mark is repeated, then the two characters are treated as a single instance of the character, not as string delimiters, for example, "Jean's book."

Localization file sections

Different files have different types of sections, and different types of sections have different entry keywords.

This section contains specific information about the sections that are common to all localization files.

Table B-1 describes where to find information on sections specific to particular files:

Table B-1: References for sections specific to a file

File name	See
The locales file (*locales.dat*)	"Locales file sections and entries" on page 48
Collating sequence files (*sort_order_name.srt*)	"Collating sequence file sections and entries" on page 58

Table B-2 describes sections that are common to all external localization files:

Table B-2: Standard sections in localization files

Section	Description	Example

File format section	This section is optional.
	If used, it has the form:
	`[file format]`
	`version = version_number`
	`list_separator = list_separator_char`
	`escape = escape_char`
	where:
	• *version_number* is a version number.
	• *list_separator_char* is the list separator character to use for the file.
	• *escape_char* is the escape character to use for the file. If not specified, "list_separator" defaults to "," (comma), and "escape" defaults to "\" (backslash).
		`[file format]`
		`version = 1`
		`list_separator =,`
		`escape = \`

Copyrightsection	This section is optional.
	If used, it has the form:
	`[copyright]`
	`copyright = "copyright_statement"`
	where *copyright_statement* is a character string.
		`[copyright]`
		`copyright = "Copyright\ Excellent Products, Inc."`

Example localization file

The partial collating sequence file included in this section illustrates some of the syntax rules discussed in "Localization file syntax rules" on page 79.

When looking at the file, please note the following:

- The first line defines the comment character as a semicolon. Any subsequent lines or phrases beginning with a semicolon are comments.

- The second line, [sortorder], is a heading for the sortorder section. Entries in this section describe and define the collating sequence. This file does not contain copyright and file format sections, which are optional.
- The list separator for the file is a comma (the default).
- The escape character for the file is a backslash (the default).
- Values that include spaces begin and end with quotation marks, such as the value for "description =".

Note The ellipsis "..." indicates deletion of actual file contents.

```
; semi-colon is the comment character
[sortorder]
;-------------------------------------------------------------------
; Overview
; --------
; Case-sensitive sort order based on the ISO 8859-1 code set.
; Uppercase characters sort before lowercase counterparts.
;
; Ligatures and sort doubles
; --------------------------
; AE, ae ligatures
; German sharp-s ligature with "ss"
;
; Sort order
; ----------
; 1. non-alphanumeric characters in binary order
; 2. numeric digits
; 3. alphabetic characters used in English, French, German
; 4. Alphabetic characters not used in English, French, German
;
; Format
; ------
; Default formatting values. There is no [file format] section.
;-------------------------------------------------------------------class = 0x01
id = 0x33
menuname = "Case-sensitive dictionary sort order"
name = dictionary
charset = iso_1
description = "Dictionary sort order for use with English,\ French and German.
ISO 8859-1,case sensitive."
;
; Ligatures for English, French, German
lig = 0xC6, after AE
```

```
lig = 0xE6, after ae
lig = 0xDF, after ss
;
; Control characters
char = 0x01(SOH) start of heading
...
char = 0x1F;(US) unit separator
;
; All non-alphanumeric characters, including punctuation,sorted
; by numerical ordering
char = 0x20;( ) space
...
char = 0xF7;division sign
;
; Digits
char = 0x30;(0) digit zero
...
char = 0x39;(9) digit nine
;
; Latin alphabet
char = 0x41,0x61,0xC0,0xE0,0xC1,0xE1,0xC2,0xE2,0xC3,0xE3,0xC4,
0xE4,0xC5,0xE5
; letter A, a, A-grave, a-grave, A-acute, a-acute, A-circumflex,
; a-circumflex, A-tilde, a-tilde, A-diaeresis, a-diaeresis,
; A-ring, a-ring
...
char = 0x5A,0x7A;letter Z,z
;
; Alphabetic characters not used in English, French, German
char = 0xD0,0xF0;Icelandic letter Eth, eth
...
```

Glossary

case-sensitive
When applied to a collating sequence, it means that the collating sequence distinguishes between uppercase and lowercase characters.

character
A member in a set of elements that represents data in a native language, such as “e,” “ë,” “5,” or “¿.”

character set
A finite set of characters and glyphs that can include letters, ideographs, digits, symbols, and control functions. See also *single-byte character set* and *multibyte character set*.

coded character set
A character set in which each character is assigned a numeric code value. Also called a *code page*.

coded character set conversion
Changing the encoding of characters from one set of numeric codes to another.

When clients and servers use different character sets, coded character set conversion them to interpret data the same way.

collating sequence
The order in which a system sorts text.

digraph
See *ligature*.

encoding
For character sets, the unique identification of each character with a numeric code.

glyph
The graphic representation of a character. For example, the character “f” can be represented by the glyph “f” or “*f*.”

ideograph
A character or symbol that represents an idea, such as those used in written Chinese and Japanese.

internationalization
The process of enabling an application to support multiple languages and cultural conventions. An internationalized application uses the language and cultural conventions appropriate to the geographic area in which it is running.

ligature
A single character that is sorted as multiple characters. For example, “→” is sorted as “AE,” and “β,” sorted as “ss.”

locale — 1. A specific geographic or national language region. 2. A collection of information related to a specific geographic or national language region.

locales file — A Sybase-specific file that maps locale names to languages, character sets, and collating sequences. Open Client and Open Server products examine the locales file when loading localization information.

locales structure (CS_LOCALE) — A CS-Library structure that is used to define custom localization values in Client-Library and Server-Library applications. The CS-Library routines cs_loc_alloc and cs_loc_drop allocate and drop a locale structure. The CS-Library routine cs_locale loads a locale structure with information.

localization — The process of setting up an application to execute using a specific language and related cultural conventions.

multibyte character set — A character set that includes characters that are encoded using more than one byte, such as EUC JIS and Shift-JIS. A multibyte character set can include characters of varying widths.

single-byte character set — A character set in which all characters are encoded using a single byte.

sort double — In a collating sequence, a pair of characters that is sorted as a single character. For example, “ch” in Spanish.

sort order — See *collating sequence*.

Unicode — A universal, 16-bit encoded character set, defined by the Unicode Standard. Unicode version 1.1 is code-for-code identical to ISO 10646, the international standard universal character set.

UTF-8 — An encoding that is the UCS Transformation Format, 8-bit form. It uses multibyte characters up to 4 bytes long.

UTF-16 — An encoding that is the UCS Transformation Format, 16-bit form. In UTF-16, each UCS-2 code value represents itself, where all of the characters currently defined are 2 bytes long. Code values beyond the BMP (Basic Multilingual Plane: 0..0xFFFF) are represented using pairs of special codes called surrogate pairs.

Index

B

C

D

E

F

I

K

L

M

O

P

R

S

T

U

V