Programmer’s Supplement

Open Client™ and Open Server™

12.5.1

UNIX

DOCUMENT ID: DC35456-01-1251-02

LAST REVISED: April 2004

Contents

About This Book

The Sybase® Open Client™ and Open Server™ products are a set of programming interfaces that allow applications and data of any type to be used together. They include:

- Open Client DB-Library™/C
- Open Client Client-Library™/C
- Open Server Server-Library/C
- Open Client Embedded SQL™/C
- Open Client Embedded SQL/COBOL

Each of these products has its own reference manual that describes it in detail. The purpose of this book is to serve as a supplement to the product manuals. It describes the platform-related issues for all the Open Client and Open Server products.

The following UNIX platforms are covered:

- HP Tru64 UNIX
- HP 9000 Series HP-UX
- IBM RISC System/6000 AIX
- Linux
- Silicon Graphics IRIX
- Sun Solaris 2.x (SPARC)

Audience

This manual is written for programmers who use the Open Client and Open Server products listed above.

How to use this book

This supplement contains material for all the Open Client and Open Server products running on the UNIX operating system. Each product, such as Open Client Client-Library or Open Server, is covered in its own chapter. The chapters such discuss issues as:

- Building an executable

- Information on the online sample programs, including their locations and what they do

The appendixes contain the following:

- Reference pages that detail the syntax, parameters, and qualifiers for the commands and utilities relevant to Open Client and Open Server
- Information about the environment variables you need to set so that you can build and run your applications

Related documents

Each Open Client and Open Server product has its own set of user documentation. Table 1 lists the products and their related documents:

Table 1: Product documentation list

Product	Related Documentation
Client-Library	Open Client *Client-Library/C Reference Manual* Open Client and Open Server *Common Libraries Reference Manual* Open Client *Client-Library/C Programmer's Manual*
DB-Library	Open Client *DB-Library/C Reference Manual*
Server-Library	Open Server *Server-Library Reference Manual* Open Client and Open Server *Common Libraries Reference Manual* Open Client *Client-Library Reference Manual*
ESQL/C	Embedded SQL/C *Programmer's Manual* Embedded SQL *Programmer's Reference*
ESQL/COBOL	Embedded SQL/COBOL *Programmer's Manual* Embedded SQL *Programmer's Reference*

See your installation guide for information on installation, directory structure, and logical names.

See the Open Client and Open Server *Configuration Guide* for UNIX for information on how to:

- Set up your environment so that Open Client applications and servers can communicate
- Localize Sybase applications

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD, and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library 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 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
3. Select a product.
4. Specify a time frame and click Go.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The syntax conventions used in this manual are as follows:

Table 2: Syntax conventions

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

The following examples illustrate the syntax conventions described above.

Use of *vertical bars* means to choose one and only one option within the *curly braces:*

```
{red | yellow | blue}
```

Use of *commas* means to choose one or more options within the *curly braces*. If you choose more than one, separate your choices with commas.

```
{cash, check, credit}
```

Brackets indicate optional parameters. You do not have to choose any of the items in square brackets.

One item in square brackets means you can omit it entirely.

```
[anchovies]
```

Use of *vertical bars* means you can choose none or only one within the *square brackets*.

```
[beans | rice | sweet_potatoes]
```

Commas within *square brackets* means you can choose none, one, or more options. If you choose more than one, separate your choices with commas.

```
[extra_cheese, avocados, sour_cream]
```

Syntax followed by *Ellipsis (...)* indicates that you can repeat the last unit as many times as you like. In the following syntax example, one or more pairs of program names and extensions can be listed between the square brackets:

```
cpre -L program[.ext] [program[.ext]]...
```

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, ask a designated person at your site to contact Sybase Technical Support.

CHAPTER 1

Open Client Client-Library/C

Open Client Client-Library is a collection of routines for use in writing client applications. Client-Library includes routines that send commands to a server and other routines that process the results of those commands. Still other routines set application properties, handle error conditions, and provide a variety of information about an application's interaction with a server.

CS-Library, which is included with Open Client, is a collection of utility routines that can be used in writing both Open Client and Open Server applications. All Client-Library applications include at least one call to CS-Library, because Client-Library routines use a structure which is allocated in CS-Library.

This chapter covers the following topics:

Note Refer to the release bulletin for additional information about Open Client products and how they behave on your platform.

General instructions

To run the Client-Library sample programs, you must:

- Be able to access an Adaptive Server. See the descriptions of the individual samples for the required Adaptive Server version level.
- Set the following environment variables, which are described in Appendix B, "Environment Variables":
  - SYBASE

    - DSQUERY
    - SYBPLATFORM
    - Platform-specific library path variable
- Be able to connect to an Adaptive Server™. Refer to the Open Client and Open Server *Configuration Guide* for your platform for information about connecting to an Adaptive Server.
- Read the *README* file for complete instructions on running the sample programs.

Building a Client-Library executable

This section discusses the libraries and compile and link lines needed to build Client-Library applications, including multithreaded applications.

Table 1-1 lists the libraries that you need to include to take full advantage of all Client-Library capabilities in a non-threaded environment.

Table 1-1: Platform-specific libraries

Platform	Required libraries
All platforms	*libct* – >index-marker text="Libraries:libct; libct "> Client-Library (Sybase) *libcs* – CS-Library (Sybase) *libtcl* – Transport Control Layer (Sybase internal) *libcomn* – An internal shared utility library (Sybase internal) *libintl* – Internationalization support library (Sybase internal))

Native thread support

Client-Library version includes thread-safe libraries which allows developers to create multithreaded applications using Posix threads.

Refer to “Compile-and-link lines for multithreaded applications” on page 7 for proper syntax and examples.

Table 1-2 lists the libraries that you need to include to take advantage of all Client-Library capabilities for multithreaded support.

Table 1-2: Platform-specific libraries for multithreaded support

Platform	Required libraries
All platforms	*libct_r* – Client-Library (Sybase) *libcs_r* – CS-Library (Sybase) *libintl_r* – Internationalization support library (Sybase internal) *libm* – Standard UNIX math library (system)
Sun Solaris	*libpthread* – Thread library (system) *socket* – Socket network library (system) *libnsl* – A network library (system) *libdl* – Dynamic loader library (system) *libtcl_r* – Transport Control Layer (Sybase internal) *libcomn_r* – Internal shared utility library (Sybase internal)
HP 9000(8xx)	*libcl* – HP Transport Control Layer (system) *libBSD* – The BSD library (system) *libc_r* – C reentrant library *libndbm* – (system) *libtcl_r* – Transport Control Layer (Sybase internal) *libcomn_r* – Internal shared utility library (Sybase internal)
IBM RS/6000	*libc_r* – C reentrant library *libpthreads* – Thread library (system) *libtcl_r* – Transport Control Layer (Sybase internal) *libcomn_r* – Internal shared utility library (Sybase internal)
Linux	*libtcl_r* – Transport Control Layer (Sybase internal) *libc_r* – C reentrant library *libpthreads* – Thread library (system) *libtcl_r* – Transport Control Layer (Sybase internal) *libcomn_r* – Internal shared utility library (Sybase internal

Kerberos support

Client-Library version 11.1 and later supports Kerberos security features for applications that desire a high level of security when communicating over a network. By installing the required Kerberos software, and performing the appropriate configuration tasks, your Client-Library applications can take advantage of the following Kerberos security features that are supported in this version:

- Network authentication
- Mutual authentication
- Out-of-sequence authentication
- Replay detection

- Confidentiality
- Integrity

Note There is no Kerberos support for HP Tru64 UNIX or SGI platforms at this time. Refer to your release bulletin for the latest information regarding additional support.

To develop and run Client-Library applications that take advantage of Kerberos features, perform the tasks listed in Table 1-3.

Table 1-3: Required tasks for Kerberos support

Tasks	For more information
Install the Kerberos software on your system.	Refer to your Kerberos documentation and the Open Server *Configuration Guide* for UNIX for instructions.
Configure the security section of the *libtcl.cfg* configuration file.	See the Open Client and Open Server *Configuration Guide* for UNIX.
Log in to the Kerberos security environment with the Kerberos kinit utility, before running your Client-Library application.	Refer to your Kerberos documentation.
Set the environment variable to the credential cache directory location.: • For Cybersafe, CSFC5CCNAME • For MIT, KRB5CCNAME	Refer to your Kerberos documentation. Default credential cache directory location varies by platform.
Set the desired security features using ct_con_props or use the default credentials by not setting ct_con_props.	See the Open Client *Client-Library/C Reference Manual*. Use CS_SUPPORTED action type in ct_con_props and ct_config to determine if a security feature is supported.

Compile and link lines

Client-Library and Server-Library dynamically link directory drivers and security drivers. This means that you must not explicitly link the following associated libraries with your applications:

- Sybase Net-Library drivers (linker option -linsck for HP-UX, and IBM RS/6000 or -ltli for Sun Solaris 2.x)

- Sybase directory or security drivers (linker options -lldap and -lskrb)

Compile-and-link lines for non-threaded applications

The following tables list the general forms of the commands for compiling and linking non-threaded Client-Library applications on Sybase supported platforms running on UNIX. (Also, refer to the *Makefile* in the *$SYBASE/$SYBASE_OCS/sample/ctlibrary* directory for compile and link information.)

Table 1-4 shows commands for compiling and linking Client-Library applications using static libraries.

Table 1-4: Static link-and-compile commands for Client-Library

Platform	Command
Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib APP_FILES OCS_LIBS -Bstatic -lcs -ltcl -lcomn -lintl -Bdynamic -lnsl -ldl -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib APP_FILES OCS_LIBS -lcs -ltcl -lcomn -lintl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib APP_FILES -Wl,a,archive OCS_LIBS -lcs -ltcl -lcomn -lintl -Wl,-a,default -lcl -lm -lBSD -ldld -Wl,-E,+s -o program`
SGI	`cc -o [-n32 |-n64] -mips3 -I$SYBASE/$SYBASE_OCS/include` *`APP_FILES`* `-L$SYBASE/$SYBASE_OCS/lib -Bstatic` *`OCS_LIBS`* `-lcs -ltd -lcomn -lintl -Bdynamic -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib` *`APP_FILES OCS_LIBS`* `-lcs -ltcl -lcomn -lintl -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib` *`APP_FILES OCS_LIBS`* `-lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Table 1-5 shows commands for compiling and linking Client-Library applications using debug libraries.

Table 1-5: Debug link-and-compile commands for Client-Library

Platform	Command
Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib \ -g` *`APP_FILES OCS_LIBS`* `-lcs -ltcl -lcomn -lintl -Bdynamic -lnsl -ldl -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib -g` *`APP_FILES`* `\` *`OCS_LIBS`* `-lcs -ltcl -lcomn -lintl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib -g` *`APP_FILES`* `\ -Wl,-a,archive` *`OCS_LIBS`* `-lcs -ltcl -lcomn -lintl \-Wl,-a,default -lcl -lm -lBSD -ldld -Wl,-E,+s -o program`
SGI	`cc -g [-n32 |-n64] -mips3 -I$SYBASE/$SYBASE_OCS/include \ -L$SYBASE/$SYBASE_OCS/devlib` *`APP_FILES OCS_LIBS`* `-lcs -ltcl -linsck \ -lcomn -lintl -lm -o program`
HP Tru64 UNIX	`cc -g -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib` *`APP_FILES OCS_LIBS`* `\-lcs -ltcl -oldstyle_liblookup -lcomn -lintl -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib` *`APP_FILES OCS_LIBS`* `-lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Table 1-6 shows commands for compiling and linking Client-Library applications using shareable libraries (with dynamic drivers).

Table 1-6: Shareable link-and-compile commands for Client-Library

Platform	Command

Sun Solaris 2.x	`/opt/SUNWspro/bin/cI$SYBASE/$SYBASE_OCS/include`
	`-L$SYBASE/$SYBASE_OCS/lib \`
	`-R$SYBASE/$SYBASE_OCS/lib` *APP_FILES OCS_LIBS*
	`-lcs -lcomn \ -ltcl -lintl -lnsl -ldl -lm`
	`-o program`

HP 9000(8xx)	`cc -I$SYBASE/$SYBASE_OCS/include`
	`-L$SYBASE/$SYBASE_OCS/lib` *APP_FILES OCS_LIBS*
	`-lcs -lcomn \-ltcl -lintl -linsck -Wl -lcl -lm`
	`-lBSD -o program`

SGI	`cc [-n32 |-n64] -mips3`
	`-I$SYBASE/$SYBASE_OCS/include \`
	`-L$SYBASE/$SYBASE_OCS/lib` *APP_FILES OCS_LIBS*
	`-lcs -ltcl -linsck \ -lcomn -lintl -lm -o program`

HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include`
	`-L$SYBASE/$SYBASE_OCS/lib` *APP_FILES OCS_LIBS*
	`-lcs -ltcl \ -oldstyle_liblookup -lcomn -lintl`
	`-lm -o program`

Compile-and-link lines for multithreaded applications

Table 1-7 shows commands for compiling and linking Client-Library applications with libraries to take advantage of thread-safe support.

Table 1-7: Thread-safe link-and-compile commands for Client-Library

Platform	Command

Sun Solaris 2.3 and later	`/opt/SUNWspro/bin/cc`
	`-I$SYBASE/$SYBASE_OCS/$SYNASE_OCS/include`
	`-L$SYBASE/$SYBASE_OCS/$SYBASE_OCS/lib -g \`
	`-D_REENTRANT` *APP_FILES OCS_LIBS* `-lcs_r -ltcl_r`
	`-lcomn_r \-lintl_r -Bdynamic -lnsl -ldl -lpthread`
	`-lthread -lm -o program`

IBM RS/6000	`xlc_r4 -`
	`I$SYBASE/$SYBASE_OCS/$SYBASE/$SYBASE_OCS/`
	`$SYBASE_OCS/include -`
	`L$SYBASE/$SYBASE_OCS/$SYBASE/$SYBASE_OCS`
	`$SYBASE/$SYBASE_OCS/$SYBASE_OCS/lib -g`
	`-D_THREAD_SAFE \`*APP_FILES OCS_LIBS* `-lcs_r`
	`-ltcl_r -lcomn_r -lintl_r \`
	`-lxdsxom -lpthread -lm -o program`

Platform	Command
HP 9000(8xx)	`cc -I$SYBASE/$SYBASE_OCS/$SYBASE/$SYBASE_OCS/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/$SYBASE_OCS/lib -g -D_THREAD_SAFE \-D_REENTRANT -Ae` *`APP_FILES OCS_LIBS`* `-lcs_r \-ltcl_r -lcomn_r -lintl_r -Wl, -a,default -lcl -lm -lBSD \ -lpthread -lc_r -ldld -Wl,-E,+s -o program`
HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib -threads` *`APP_FILES`* `\`*`OCS_LIBS`* `-lcs_r -ltcl_r -lcomn_r -lintl_r -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/vlib` *`APP_FILES OCS_LIBS`*`-lcs_r -lsybtcl_r -lcomn_r -lintl_r -rdynamic -ldl -lpthread -lnsl -lm -o program`

In the link-and-compile lines listed in Table 1-4 through Table 1-7:

- *APP_FILES* represents the source (.c) or object (.o) files for your application.
- *OCS_LIBS* represents the linker options to link in the Open Client and Open Server libraries that your code calls. These options can be specified by any or all of the following linker options, in the order shown:
  - For non-threaded applications:

    -lsrv (for Server-Library routines)

    -lblk (for Bulk-Library routines)

    -lct (for Client-Library routines)
  - For threaded applications:

    -lsrv_r (for Server-Library routines)

    -lblk_r (for Bulk-Library routines)

    -lct_r (for Client-Library routines)

For HP-UX system users:

- The option -W1,-a,archive causes the linker to statically link the Sybase libraries. By not specifying this option, Client-Library uses shared versions of the Sybase libraries are used. When using shared libraries, the SH_LIB_PATH environment variable must include *$SYBASE/$SYBASE_OCS/lib* at runtime, and the application user must have read and execute permission on the libraries in *$SYBASE/$SYBASE_OCS/lib*.
- HP-UX will not use the SH_LIB_PATH environment variable at runtime unless the application is linked with the +s linker option. You must use the +s linker options so that the system will be able to find Sybase libraries at runtime. -E is required to prevent undefined-symbol errors when driver libraries are loaded at runtime. See the HP-UX ld man page for more information.

For SGI users:

- Use the -n32 option for 32-bit machines, and -n64 for 64-bit machines.

Note You must set the environment variable LD_LIBRARY_PATH to *$SYBASE/$SYBASE_OCS/lib* to run programs linked with shareable (dynamic) libraries. If you are running in debug mode, set LD_LIBRARY_PATH to *$SYBASE/$SYBASE_OCS/devlib* to run the program.

For HP Tru64 UNIX users:

The library file extensions for HP Tru64 UNIX are *.a* for static libraries and *.so* for sharable libraries. Use the following general command form to compile and link a Client-Library application:

Nonthreaded non-reentrant libraries

- Optimized libraries – nonthreaded, non-reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include program.c \
 $SYBASE/$SYBASE_OCS/lib/libct.a \
 $SYBASE/$SYBASE_OCS/lib/libcs.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn.a \
 $SYBASE/$SYBASE_OCS/lib/libintl.a \
 -lm -o program
```

- Debug libraries – nonthreaded, non-reentrant:

```
cc -g -I$SYBASE/$SYBASE_OCS/include \
 -L$SYBASE/$SYBASE_OCS/devlib program.c \
 -lct -lcs -ltcl -oldstyle_liblookup \
 -lcomn -lintl \
 -lm -o program
```

- Shareable libraries – nonthreaded, non-reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include
-L$SYBASE/$SYBASE_OCS/lib program.c \
 -lct -lcs -ltcl -oldstyle_liblookup \
 -lcomn -lintl \
 -lm -o program
```

Threaded reentrant libraries

- Optimized libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include -threads program.c
\
 $SYBASE/$SYBASE_OCS/lib/libct_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcs_r.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn_r.a \
 $SYBASE/$SYBASE_OCS/lib/libintl_r.a \
 -lm -o program
```

- Debug libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include -g -threads
program.c \
 -L$SYBASE/$SYBASE_OCS/devlib \
 -oldstyle_liblookup -lct_r -lcs_r \
 -ltcl_r -lintl_r \
 -lm -o program
```

- Shareable libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include -threads program.c
\
 -oldstyle_liblookup \
 -lct_r -lcs_r -ltcl_r \
 -lcomn_r -lintl_r \
 -lm -o program
```

Compile-and-link lines for Kerberos supported applications

The Sybase driver for Kerberos is a dynamically-loaded shared library. When the driver is loaded, it attempts to dynamically load a Kerberos *GSS* library. This needs to be in the search path that the dynamic loader uses. Due to constraints in the implementation of the Sybase driver, only re-entrant libraries are supported when using Kerberos.

Bulk copy routines

If you plan to use bulk copy routines, link in the *libblk* bulk copy library. If you plan to use bulk-copy routines in a threaded applications, link in the *libblk_r* bulk-copy library.

To link in the bulk-copy library:

- In non-threaded applications, add -lblk before -lct on the link line.
- In multithreaded applications, add -lblk_r before -lct_r on the link line.

See the Open Client and Open Server *Common Libraries Reference Manual* for more information on bulk copying.

Performance considerations

Linking with shared libraries results in a smaller executable and takes less time than linking with static libraries. However, executables linked with shared libraries may have a slower start-up time than those linked with static libraries. Also, unlike static libraries, the shared libraries must be available at runtime.

The type of library that provides the best performance is determined by your individual site requirements.

Header files

Include the *ctpublic.h* header file in all Client-Library application source files. Other necessary header files are nested in *ctpublic.h*. If Bulk-Library is used, include *bkpublic.h* instead of *ctpublic.h*.

See the Open Client *Client-Library/C Reference Manual* for more information on header files.

Using Client-Library sample programs

Sample programs are included online with Client-Library to demonstrate typical uses for Client-Library routines. Use the following information to compile and run the samples.

Some sample programs use the sample databases supplied with Adaptive Server. Refer to your installation guide for information on installing the sample databases. The requirements section for each sample lists the database you need, if any.

Note New sample programs are included for the Open Client 12.5 wide table and unichar capabilities. These programs have *uni_* or *wide_* prefixes and are described in "Sample program summaries." Some of these programs require the unipubs database.

Makefile and sample programs

In order to use the *makefile* to build sample programs on all platforms, you must set the SYBPLATFORM environment variable correctly for the compiler you are using. For the environment variables and library path refer to Table B-1 on page 114.

Building applications using shared libraries

To build applications on the IBM platform using shared libraries, set up your makefile to use a link statement like this (where export files are named **.exp*):

```
xlc_r -g -D_THREAD_SAFE  -I. \
    -I$SYBASE/$SYBASE_OCS/include \
    -DDEBUG -Dnthread_rs6000=1 multthrd.c \
    thrdutil.o thrdfunc.o \
    -bI:$SYBASE/$SYBASE_OCS/libct_r.exp \
    -bI:$SYBASE/$SYBASE_OCS/libcs_r.exp \
    -bI:$SYBASE/$SYBASE_OCS/libcomn_r.exp \
    -lm -o multthrd
```

In order to execute such an application, make sure that:

- The shared libraries are in the directory *$SYBASE/$SYBASE_OCS/lib.*
- The LIBPATH environment variable is set to:

  $SYBASE/$SYBASE_OCS/lib:/lib
- The configuration file *$SYBASE/$SYBASE_OCS/config/libtcl.cfg* specifies the correct driver.

Purpose of the sample programs

The sample programs demonstrate specific Client-Library functionality. These programs are designed as guides for application programmers, not as Client-Library training aids. Read the descriptions at the top of each source file, and examine the source code prior to using the sample programs.

These simplified programs are not intended for use in a production environment. Production-quality programs require additional code to handle errors and special cases.

The sybopts.sh script and building applications

The *sybopts.sh* script is included with the sample programs and helps you build Open Client and Open Server applications for your platform by reading the SYBPLATFORM environment variable:

```
sybopts.sh <args>
```

where *args* can be:

- compile – returns the compiler command and platform-specific compile flags.
- comlibs – returns the list of required Sybase libraries that must be linked with the application.
- syslibs – returns the list of required non-Sybase system libraries that must be linked with the application.

Location

The sample programs are located in the following directory:

```
$SYBASE/sample/ctlibrary
```

This directory includes:

- Online source code for the sample programs.
- Data files for the samples.
- The *makefile* provided to build the samples. Use the *makefile* as a starting point for your own Client-Library applications.
- The samples header file, *example.h.*

- The *README* file containing instructions for building, executing, and testing the samples.

Note Before compiling and running the sample programs, copy the contents of *$SYBASE/sample/ctlibrary* into a “working” directory, where you can freely experiment with the sample programs without affecting the integrity of the original files.

Header file

All of the sample programs reference the sample header file, *example.h*, the contents of which are as follows:

```
/*
 ** example.h
 **
 ** This is the header file that goes with the
 ** Sybase Client-Library example programs.
 **
 **
 */

 /*
 ** Define symbolic names, constants, and macros
 */
 #define EX_MAXSTRINGLEN     255
 #define EX_BUFSIZE          1024
 #define EX_CTLIB_VERSION    CS_VERSION_125
#define EX_BLK_VERSION    BLK_VERSION_125
#define EX_ERROR_OUT        stderr

 /*
 ** exit status values
 */
 #define  EX_EXIT_SUCCEED 0
 #define  EX_EXIT_FAIL 1

 /*
 ** Define global variables used in all sample
 ** programs
 */
 #define EX_SERVER           NULL/* use DSQUERY
                                  env var */
```

```
#define EX_USERNAME         "user"
#define EX_PASSWORD         "server_password"
```

The sample programs make use of the define statements in *example.h* as illustrated in the following fragments:

```
CS_CHAR *Ex_username = EX_USERNAME;
 CS_CHAR *Ex_password = EX_PASSWORD;

 /*
 ** If a user name is defined, set the
 ** CS_USERNAME property.
 */
 if (retcode == CS_SUCCEED && Ex_username != NULL)
 {
     if ((retcode = ct_con_props(*connection,
         CS_SET, CS_USERNAME, Ex_username,
         CS_NULLTERM, NULL)) != CS_SUCCEED)
     {
         ex_error("ct_con_props(username) failed");
     }
 }

 /*
 ** If a password is defined, set the
 ** CS_PASSWORD property.
 */
 if (retcode == CS_SUCCEED && Ex_password != NULL)
 {
     if ((retcode = ct_con_props(*connection,
         CS_SET, CS_PASSWORD, Ex_password,
         CS_NULLTERM, NULL)) != CS_SUCCEED)
     {
         ex_error("ct_con_props(password) failed");
     }
 }
```

Edits for these lines in *example.h* are described in the following sections.

EX_USERNAME

EX_USERNAME is defined in *example.h* as “sa.” Before running the sample programs, you must edit *example.h* and change “sa” to your server login name.

EX_PASSWORD

EX_PASSWORD is defined in *example.h* as “ ”. Before running the sample programs, you may want to edit *example.h* and change “ ” to your server password.

You have three options regarding EX_PASSWORD. Choose the one that best meets your needs:

- *Option 1* – Change your server password to “server_password” while you are running the samples. This creates the possibility of a security breach, because while your password is set to this published value, an unauthorized person might take the opportunity to log in to the server as you. If this is a problem, choose one of the other methods of handling passwords for the sample programs.
- *Option 2* – In *example.h*, change the string “server_password” to your own server password. Use the operating system’s protection mechanisms to prevent others from accessing the header file while you are using it. When you are finished with the samples, edit the line so that it again says “server_password.”
- *Option 3* – In the sample programs, modify the ct_con_props code that sets the server password and substitute your own code to prompt samples users for their server passwords. (Because this code is platform-specific, Sybase does not supply it.)

Utility routines for the sample programs

The *exutils.c* file contains utility routines that are used by all other Client-Library sample programs. It demonstrates how an application can hide some of the implementation details of Client-Library from a higher-level program.

For more information about these routines, see the leading comments in the sample source file.

Sample program summaries

The following sample programs are included with your software.

uni_blktxt.c

The *uni_blktxt.c* sample program uses the bulk-copy routines to copy static data to a server table. There are three rows of data that are bound to program variables and then sent to the server as a batch. The rows are again sent using blk_textxfer to send the text data. For more information about this sample program, see the leading comments in the sample source file.

Note This example requires a SQL Server version 4.9.1 or later.

---

compute.c

The *compute.c* sample program demonstrates processing compute results. It sends a canned query to the server using a language command. It processes the results using the standard ct_results while loop. It binds the column values to program variables. It then fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

```
select type, price from titles
 where type like "%cook"
 order by type, price
 compute sum(price) by type
 compute sum(price)
```

This query returns both regular rows and compute rows. The compute rows are generated by the two compute clauses. The first compute clause, "compute sum(price) by type," generates a compute row each time the value of type changes. The second compute clause "compute sum(price)," generates one compute row, which is the last to be returned.

For more information about this sample program, see the leading comments in the sample source file.

Note This sample requires the pubs2 database.

---

uni_csr_disp.c

The *uni_csr_disp.c* example program demonstrates using a read-only cursor. It opens a cursor with a canned query. It processes the results using the standard ct_results while loop. It binds the column values to program variables. Then, it fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

```
select au_fname, au_lname, postalcode
 from authors
```

For more information about this sample program, see the leading comments in the example source file.

Note This sample requires a SQL Server version 10.0 or later, and the pubs2 database.

ex_alib.c and *ex_amain.c*

This sample program contains two files, *ex_alib.c* and *ex_amain.c*, which demonstrate how to write an asynchronous layer on top of Client-Library. It uses hooks provided by Client-Library to allow seamless polling and use of Client-Library's completion callbacks.

The sample program is composed of two files:

- *ex_alib.c* contains the source code to the library portion of the example. It is meant to be part of a library interface which supports asynchronous calls. This module provides a means of sending a query to and retrieving the results from a server within one asynchronous operation.
- *ex_amain.c* contains the source code to the main program that uses the services provided by *ex_alib.c*.

For more information about this example program, see the leading comments in the example source file and the *EX_AREAD.ME* file.

exconfig.c

The *exconfig.c* sample program demonstrates how Client-Library application properties can be configured externally.

This sample requires you to edit the default runtime configuration file, *\config\ocs.cfg*, located in the Sybase installation directory. The example sets the CS_CONFIG_BY_SERVERNAME Client-Library property and calls ct_connect with a *server_name* parameter set to "server1." In response, Client-Library looks for a [Server1] section in the external configuration file. To run the example, create *\config\ocs.cfg* in the Sybase installation directory (if necessary) and add the section:

```
[server1]
```

```
CS_SERVERNAME = real_server_name
```

where *real_server_name* is the name of the server that you want to connect to.

For more information on how Client-Library uses external configuration files, see the topics page “Using the Runtime Configuration File” in the Open Client *Client-Library/C Reference Manual*.

firstapp.c

The *firstapp.c* sample program is an introductory example that connects to the server, sends a select query, and prints the rows. This example program is described in the Open Client *Client-Library/C Programmer’s Guide*.

getsend.c

The *getsend.c* sample program demonstrates how to retrieve and update text data from a table containing text along with other datatypes. The process demonstrated can also be used for retrieving and updating image data. For more information about this sample program, see the leading comments in the example source file.

Note This example requires SQL Server version 10.0 or later.

i18n.c

The *i18n.c* example program demonstrates some of the international features available in Client-Library, including:

- Localized error messages
- User-defined bind types

For more information about this program, see the leading comments in the example source file.

multthrd.c and *thrdfunc.c*

This sample program contains two files, *multthrd.c* and *thrdfunc.c*, which demonstrate a multithreaded Client-Library application. The following information is contained in the two files:

- *multthrd.c* contains the source code that spawns five threads. Each thread processes a cursor or a regular query. The main thread waits for the other threads to complete query processing and then terminates.
- *thrdfunc.c* contains platform specific information that determines which thread and synchronization routines the example uses for execution depending on your platform.

For more information about this program, see the leading comments in the example source file.

This sample cannot run if the platform does not support a complete POSIX thread implementation. You must set the SYBPLATFORM environment variable described in Appendix B, "Environment Variables."

Note This example requires SQL Server version 10.0 or later.

rpc.c

The RPC command example program, *rpc.c*, sends an RPC command to a server and processes the results. For more information about this example program, see the leading comments in the example source file.

Note This example requires a SQL Server version 4.9.1 or later.

secct.c

The *secct.c* sample program demonstrates how to use network-based security features in a Client-Library application.

For this sample to execute, Kerberos must be installed and running on your machine. You must also connect to a server that supports network-based security, such as ASE or the *secsrv.c* Open Server sample program.

For more information about this example program, see the leading comments in the example source file. For more information about network security services, refer to the Open Client and Open Server *Configuration Guide* for UNIX.

uni_blktxt.c

The *uni-blktxt.c* sample program uses the bulk-copy routines to copy static data to a server table. This program has been modified for the use of the unichar and univarchar datatypes. There are three rows of data that are bound to program variables and then sent to the server as a batch. The rows are again sent using blk_textxfer to send the text data.

uni_compute.c

The *uni_compute.c* sample program demonstrates processing compute results. It is a modification of the *compute.c* sample program for the unichar and univarchar datatypes and requires the unipubs database. It sends a canned query to the server using a language command. It processes the results using the standard ct_results loop. It binds the column values to program variables. It then fetches and displays the rows in the standard ct_fetch loop.

uni_csr.c

The *uni_csr_disp.c* example program demonstrates using a read-only cursor. It is a modification of the *csr_disp.c* sample program and requires the unipubs database. It opens a cursor with a canned query. It processes the results using the standard ct_results while loop. It binds the column values to program variables. Then, it fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

```
select au_fname, au_lname, postalcode
 from authors
```

uni_firstapp.c

This is a modification of the *firstapp.c* example program for use with unichar and univarchar datatypes. It is an introductory example that connects to the server, sends a select query, and prints the rows. The *firstapp.c* program is described in the Open Client *Client-Library/C Programmer's Guide*.

uni_rpc.c

The RPC command sample program, *uni_rpc.c*, sends an RPC command to a server and processes the results. This is a modification of the *rpc.c* sample program for use with unichar and univarchar datatypes, and requires the unipubs database.

For more information about this program, see the leading comments in the example source file.

usedir.c

This sample program demonstrates Client-Library's ability to query a directory service for a list of available servers.

usedir.c searches for Sybase server entries in the default directory, as defined in the driver configuration file. If a network directory service is not being used, *usedir.c* queries the interfaces file for server entries. Then, it displays a description of each entry found, and lets the user choose a server to connect to.

For more information about this program, see the leading comments in the example source file. For more information about network directory services, refer to the Open Client and Open Server *Configuration Guide* for UNIX.

wide_*compute.c*

The *wide_compute.c* sample program demonstrates processing compute results with wide tables and larger column sizes, implemented in Open Client and Open Server version 12.5. It sends a canned query to the server using a language command. It processes the results using the standard ct_results while loop. It binds the column values to program variables. Then, it fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

```
select type, price from titles
 where type like "%cook"
 order by type, price
 compute sum(price) by type
 compute sum(price)
```

This query returns both regular rows and compute rows. The compute rows are generated by the two compute clauses. The first compute clause, "compute sum(price) by type," generates a compute row each time the value of type changes. The second compute clause "compute sum(price)," generates one compute row, which is the last to be returned.

For more information about this sample program, see the leading comments in the sample source file.

Note This sample requires the pubs2 database.

wide_curupd.c

This program uses a cursor to retrieve data from the table called "publishers" in the pubs2 database. It retrieves data row by row and prompts the user to input new values for the column state in the publishers table.

Inputs value for the input parameter (state column from the publishers table) for the UPDATE. Create a publishers3 table as shown before running the sample program:

```
use pubs2
go
drop table publishers3
go
create table publishers3 (pub_id char(4) not null,
pub_name varchar(400) null, city varchar(20) null,
state char(2) null)
go
select * into publishers3 from publishers
go
create unique index pubind on publishers3(pub_id)
```

wide_dynamic.c

This program uses a cursor to retrieve data from the table called "publishers" in the pubs2 database. It retrieves data row by row and prompts the user to input new values for the column called "state" in the publishers table.

This program uses Dynamic SQL to retrieve values from the titles table in the tempdb database. The select statement, which contains placeholders with identifiers, is sent to the server to be partially compiled and stored. Therefore, every time you call the select, you only pass new values for the key value which determines the row to be retrieved. The behavior is similar to passing input parameters to stored procedures. The program also uses cursors to retrieve rows one by one, which can be manipulated as required.

wide_rpc.c

The RPC command sample program, *rpc.c*, sends an RPC command to a server and processes the results. This is the same as the *rpc.c* program, but it uses wide tables and larger column sizes.

For more information about this program, see the leading comments in the example source file.

CHAPTER 2

Open Client DB-Library/C

Open Client DB-Library is a collection of routines you can use to write client applications. DB-Library is the predecessor to Client-Library. New functionality such as Directory and Security services support is not included with DB-Library. You must use Client-Library to take advantage of these new services.

DB-Library includes routines that send commands to a server and others that process the results of those commands. Other routines set application properties, handle error conditions, and provide a variety of information about an application's interaction with a server.

This chapter covers the following topics:

General instructions

To run DB-Library applications, including the sample programs, you must:

- Set the following environment variables, which are described in Appendix B, "Environment Variables":
    - SYBASE
    - DSQUERY
    - SYBPLATFORM
    - Platform-specific library path variable

- Be able to connect to Adaptive Server®. Refer to the Open Client and Open Server *Configuration Guide* for UNIX for information about connecting to Adaptive Server.
- Read the *README* file in each product directory under *$SYBASE/sample*. Complete instructions for running the samples are given in the *README* file.

See the following sections for descriptions and additional requirements of the individual sample programs.

Building a DB-Library executable

This section gives information on libraries, linking, and header files.

Libraries

The following table lists libraries that you should include if you want to take full advantage of all DB-Library capabilities. The first row in the table lists libraries that all platforms can use. Subsequent rows list libraries specific for each platform:

Table 2-1: Platform-specific libraries

Platform	Required libraries
All platforms	*libsybdb* – DB-Library (Sybase) *libm* – Standard UNIX math libraries (system)
Sun Solaris 2.x	*libnsl* – A network library (system)
HP 9000(8xx)	*libcl* – Transport Control Library (system) *libBSD* – The BSD library (system)

Link-and-compile lines

The following tables list the general forms of the commands for compiling and linking DB-Library applications on Sybase-supported platforms running the UNIX operating system. Table 2-2 shows the commands for compiling and linking DB-Library applications using static libraries.

Table 2-2: Static link-and-compile commands for DB-Library

Platform	Command
Sun Solaris 2.x	`cc -I$SYBASE/include -L$SYBASE/lib program.c -Bstatic -lsybdb \ -lnsl -Bdynamic -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/include -L$SYBASE/lib program.c -lsybdb -lm \ -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib program.c -lsybdb \ -Wl,-a,archive -lcl -lm -lBSD -o program`
SGI	`cc [-n32 | -n64] -mips3 -I$SYBASE/include program.c \ $SYBASE/lib/libsybdb.a -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/include program.c \ $SYBASE/lib/libsybdb.a -ldnet -lm -o program`

Table 2-3 shows the commands for compiling and linking DB-Library applications using debug libraries.

Table 2-3: Link-and-compile commands for DB-Library

Platform	Command
Sun Solaris 2.x	`cc -I$SYBASE/include -L$SYBASE/devlib program.c -lsybdb -lnsl \ -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/include -L$SYBASE/devlib program.c -lsybdb -lm \ -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/devlib program.c -lsybdb \ -linsck -Wl,-a, archive -lcl -lm -lBSD -o program`
SGI	`cc -g [-n32 | -n64] -mips3 -I$SYBASE/include -L$SYBASE/devlib \ program.c -lsybdb -lm -o program`
HP Tru64 UNIX	`cc -g -I$SYBASE/include -L$SYBASE/devlib program.c \ -lsybdb -ldnet -lm -o program`

Table 2-4 shows commands for compiling and linking DB-Library applications on platforms that support shareable libraries (with dynamic drivers).

Table 2-4: Shareable compile-and-link commands for DB-Library

Platform	Command
Sun Solaris 2.x	`cc -I$SYBASE/include -L$SYBASE/lib -R$SYBASE/lib program.c \ -lsybdb -lnsl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib program.c -lsybdb -Wl -lcl \ -lm -lBSD -o program`
SGI	`cc [-n32 | -n64] -mips3 -I$SYBASE/include -L$SYBASE/lib \ program.c -lsybdb -lm -o program`
Digital UNIX	`cc  -I$SYBASE/include -L$SYBASE/lib \ program.c -lsybdb -ldnet -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib program.c -lsybdb -ldnet \ -lm -o program`

Performance considerations

Linking with shared libraries results in a smaller executable and is faster than linking with static libraries. However, executables linked with shared libraries may be slower at start-up time than those linked with static libraries. Also, unlike static libraries, the shared libraries must be available at runtime.

The individual requirements of your site determine which type of library will provide the best performance.

Header files

The following header files are required by all DB-Library/C applications:

- *sybfront.h* – defines symbolic constants such as function return values, described in the Open Client *DB-Library/C Reference Manual*, and the exit values STDEXIT and ERREXIT. The *sybfront.h* file also includes type definitions for datatypes that can be used in program variable declaration.
- *sybdb.h* – contains additional definitions and typedefs, most of which are meant to be used only by the DB-Library/C routines. Use the contents of *sybdb.h* only as documented in the Open Client *DB-Library/C Reference Manual*.

- *syberror.h* – contains error severity values and should be included if the program refers to those values.

See the Open Client *DB-Library/C Reference Manual* for more information on header files.

Using DB-Library sample programs

Sample programs are included online with DB-Library to demonstrate typical uses for DB-Library routines.

Some sample programs use the sample databases supplied with Adaptive Server. Refer to your installation guide for information on installing the sample databases.

Purpose of the sample programs

The System 11 sample programs demonstrate specific DB-Library functionality. These programs are designed as guides for application programmers, not as DB-Library training aids. Read the descriptions at the top of each source file and examine the source code before you use the sample programs.

Note These simplified programs are not intended for use in a production environment. Production-quality programs require additional code to handle errors and special cases.

Location

The sample programs are located in the following directory:

```
$SYBASE/sample/dblibrary
```

This directory contains:

- Online source code for the sample programs
- Data files for the samples

- The samples header file, *sybdbex.h*
- The *README* file containing instructions for building, executing, and testing the samples

---

Note Before compiling and running the sample programs, copy the contents of *$SYBASE/sample/dblibrary* into a “working” directory, where you can freely experiment with the sample programs without affecting the integrity of the original files.

Header file

All of the sample programs reference the sample header file, *sybdbex.h*. The contents of *sybdbex.h* are as follows:

```
/*
 ** sybdbex.h
 **
 ** This is the header file that goes with the
 ** Sybase DB-Library example programs.
 **
 **
 */

 #define USER                 "user"
 #define PASSWORD             "server_password"
 #define LANGUAGE             "us_english"
 #define SQLBUFLEN            255
 #define ERR_CH               stderr
 #define OUT_CH               stdout
 extern void                  error();
 extern int                   err_handler();
 extern int                   msg_handler();
```

All of the samples except Example 5 contain these lines:

```
DBSETLUSER(login, USER);
DBSETLPWD(login, PASSWORD);
```

Following are descriptions of edits for the lines in *sybdbex.h*:

- USER is defined in *sybdbex.h* as “user.” Before running the sample programs, you must edit *sybdbex.h* and change “user” to your server login name.

- PASSWORD is defined in *sybdbex.h* as “server_password.” Before running the sample programs, edit *sybdbex.h* and change “server_password” to your server password. Choose one of the following options for PASSWORD:

  Option 1: Change your server password to “server_password” while you are running the samples. This creates the possibility of a security breach, because while your password is set to this published value, an unauthorized person might take the opportunity to log in to the server as you. If this is a problem, choose one of the other options.

  Option 2: In sybdbex.h, change the string “server_password” to your own server password. Use the operating system’s protection mechanisms to prevent others from accessing the header file while you are using it. When you are finished with the sample, edit the line so that it again says “server_password.”

  Option 3: In the sample programs, delete the DBSETLPWD line entirely, and substitute your own code to prompt users for their server passwords. (Because this code is platform-specific, Sybase does not supply it.)

- LANGUAGE: If your server’s language is *not* U. S. English, edit the LANGUAGE line in *sybdbex.h* so that it is the same as the server’s. Example 12 is the only sample that references LANGUAGE.

Sample program summary

The following sample programs are included with your software.

Example 1: Send queries, bind, and print results

The *example1.c* sample sends two queries to Adaptive Server in a single command batch, binds the results, and prints the returned rows of data.

Example 2: Insert data into a new table

The *example2.c* inserts data from a file into a newly created table, selects the server rows, and binds and prints the results. This sample requires a file named *datafile* (supplied). It also assumes that you have create database permission in your login database.

Example 3: Bind aggregate and compute results

The *example3.c* sample program selects information from the titles table in the pubs2 database and prints it. The sample program illustrates binding of both aggregate and compute results.

Note Access to Adaptive Server and the pubs2 database is required.

Example 4: Row buffering

The *example4.c* sample demonstrates row buffering. This program sends a query to Adaptive Server, buffers the returned rows, and allows you to examine them interactively.

Example 5: Data conversion

The *example5.c* sample illustrates dbconvert, a DB-Library/C routine that handles data conversion.

Example 6: Browse mode updates

The *example6.c* sample demonstrates browse-mode techniques. The sample program creates a table, inserts data into the table, and then updates the table using browse-mode routines. Browse mode is useful for applications that need to update data one row at a time.

Note *example6.c* requires a file named *datafile* (supplied). It creates the table *alltypes* in your default database.

Example 7: Browse mode and ad hoc queries

The *example7.c* sample uses browse-mode techniques to determine the source of result columns from ad hoc queries. Determining the source of result columns is important because a browse-mode application can only update columns that are derived from a browsable table and are not the result of a SQL expression.

This sample demonstrates how an application can determine which columns resulting from ad hoc queries can be updated using browse-mode techniques. It also prompts you for an ad hoc query. Notice how the results differ depending on whether the select query includes the keywords for browse and whether the table selected is able to be browsed.

Example 8: Making a remote procedure call (RPC)

The *example8.c* sample sends a remote procedure call, prints the result rows from the call, and prints the parameters and status returned by the remote procedure.

This sample requires you to have created the stored procedure rpctest in your default database. The comments at the top of the *example8.c* source code specify the create procedure statement necessary for creating rpctest.

Example 9: Text and image routines

The *example9.c* sample generates a random image, inserts it into a table, then selects the image and compares it to the original by following these steps:

1. insert all data into the row except the text or image value.
2. update the row, setting the value of the text or image to NULL. This step is necessary because a text or image column row that contains a null value will have a valid text pointer only if the null value was explicitly entered with the update statement.
3. select the row. You must specifically select the column that is to contain the text or image value. This step is necessary to provide the application's DBPROCESS with correct text pointer and text timestamp information. The application should throw away the data returned by this select.
4. Call dbtxtptr to retrieve the text pointer from the DBPROCESS. dbtxtptr's *column* parameter is an integer that refers to the select performed in step 3. For example, if the select is:

   ```
   select date_column, integer_column, text_column
        from bigtable
   ```

   and text_column is the name of the text column, dbtxtptr requires the *column* parameter to be passed as 3.
5. Call dbtxtimestamp to retrieve the text timestamp from the DBPROCESS. dbtxtimestamp's column parameter refers to the select performed in step 3.

6. Write the text or image value to Adaptive Server. An application can either:

   - Write the value with a single call to dbwritetext, or
   - Write the value in chunks, using dbwritetext and dbmoretext.

7. If you intend the application to make another update to this text or image value, it may want to save the new text timestamp that is returned by Adaptive Server at the conclusion of a successful dbwritetext operation. Access the new text timestamp by using dbtxtsnewval, and stored for later retrieval using dbtxtsput.

Note Access to an Adaptive Server that contains the pubs2 database is required.

Example 10: Inserting an image

The *example10.c* sample prompts you for an author ID and the name of a file containing an image, reads the image from the file, and inserts a new row containing the author ID and the image into the pubs2 database table called “au_pix.” For general information on inserting text or image values into a database table, see Example 9.

Note Access to an Adaptive Server that contains the pubs2 database is required. The author ID must be in the form “000-00-0000.” The *imagefile* file, provided with the sample code, contains an image.

Example 11: Retrieving an image

The *example11.c* sample retrieves an image from the au_pix table in the pubs2 database. The author ID you enter determines which row the program selects. After retrieving the row, this sample copies the image contained in the pic field to a file you specify.

There are two ways to retrieve a text or image value from Adaptive Server:

- This sample selects the row containing the value and processes the row using dbnextrow. After dbnextrow is called, dbdata can be used to return a pointer to the returned image.
- The other method is to use dbreadtext in conjunction with dbmoretext to read a text or image value in the form of a number of smaller chunks.

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

Note Access to Adaptive Server and the pubs2 database is required.

Example 12: International language routines

The *example12.c* sample retrieves data from the pubs2 database and prints it using a us_english format.

Note Access to Adaptive Server and the pubs2 database is required.

Example 13: Bulk copy

The bulk-copy sample program, *bulkcopy.c*, uses the bulk-copy routines to copy data from a host file into a newly created table containing several Adaptive Server datatypes.

Note Access to Adaptive Server is required. You must have create database and create table permission.

Example 14: Two-phase commit

The two-phase commit sample program, *twophase.c*, performs a simple update on two different servers. See the source code for the exact contents of the update. After you have run the sample, you can use isql on each of the servers to determine whether the update actually took place.

This sample requires that you have Adaptive Server running on two different servers, named SERVICE and PRACTICE, each containing the pubs2 database. If your servers are named differently, replace SERVICE and PRACTICE in the source code with the actual names of your servers.

Before running the sample, you need to make sure that your client can access both servers. Refer to the Open Client and Open Server *Configuration Guide* for UNIX for information about connecting to multiple instances of Adaptive Server.

Note If the PRACTICE server is on a different machine than the SERVICE server, the PRACTICE server must be able to connect to the SERVICE query port. For details, see the Open Client and Open Server *Configuration Guide* for UNIX.

CHAPTER 3

Open Server Server-Library/C

Open Server Server-Library/C is used to design servers that take advantage of the features of the client/server architecture. These Open Servers access data stored in foreign database management systems, trigger external events, and respond to Open Client applications.

The client/server architecture divides the work of computing between "clients" and "servers":

- Clients make requests of servers and process the servers' responses.
- Servers respond to requests and return data, parameters, and status information to clients.

In this architecture, an Open Client application program is a client, using the services provided by Adaptive Server and Open Server. Using Server-Library, you can create a complete, standalone server.

This chapter covers the following topics:

Topic	Page
General instructions	37
Building a Server-Library executable	38
Server-Library sample programs	44

General instructions

To run Open Server applications, including samples, you must:

- Be able to access Adaptive Server and the pubs2 sample database. Refer to your installation guide for information on installing the pubs2 database.
- Set the following environment variables, which are described in Appendix B, "Environment Variables":
    - SYBASE

- DSQUERY and DSLISTEN
- SYBPLATFORM
- Platform-specific library path variable

- Be able to connect to an Adaptive Server. Refer to the Open Client and Open Server *Configuration Guide* for UNIX for information about connecting to an Adaptive Server.

Building a Server-Library executable

This section gives information on libraries, linking, and header files.

Libraries

The following table lists the libraries that you should include if you want to take full advantage of all Server-Library capabilities. The first row in the table lists libraries that all platforms use. Subsequent rows list platform-specific libraries.

Table 3-1: Platform-specific libraries

Platform	Required libraries
All platforms	*libct* >– Client-Library (Sybase) *libcs*> – CS-Library (Sybase) *libtcl* – Transport Control Layer (Sybase internal) *libcomn* – Internal shared utility library (Sybase internal) *libintl* – Internationalization support library (Sybase internal) *libsrv* – Server-Library (Sybase) *libsybdb* – DB-Library (Sybase))

Compile-and-link line commands

The following tables list the general forms of the commands for compiling and linking Server-Library applications on Sybase-supported platforms running the UNIX operating system. The three tables include compile-and-link line commands.

Table 3-2 shows commands for compiling and linking Server-Library applications using static libraries:

Table 3-2: Static link-and-compile commands for Server-Library

Platform	Command

Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/include -L$SYBASE/lib program.c \`
	`-Bstatic -lsrv [-lsybdb | -lct] -lcs -ltcl -lcomn -lintl \`
	`-Bdynamic -lnsl -lm -o program`

IBM RS/6000	`xlc_r4 -I$SYBASE/include -L$SYBASE/lib program.c -lsrv \`
	`[-lsybdb| -lct] -lcs -lcomn -ltcl -lintl -lm -o program`

HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib program.c -Wl,-a,archive \`
	`-lsrv [-lsybdb | -lct] -lcs -lcomn -ltcl -lintl -linsck \`
	`-lcl -lm -lBSD -o program`

SGI	`cc -o [-n32 | -n64] -mips3 -I$SYBASE/include -L$SYBASE/lib \`
	`program.c -lsrv -Bstatic [-lsybdb | -lct] \`
	`-lcs -ltcl -lcomn -lintl -Bdynamic -lm -o progam`

HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include program.c\`
	`$SYBASE/$SYBASE_OCS/lib/libsrv.a \`
	`[$SYBASE/$SYBASE_OCS/lib/libsybdb.a | $SYBASE/$SYBASE_OCS/lib/libct.a] \`
	`$SYBASE/$SYBASE_OCS/lib/libcs.a $SYBASE/$SYBASE_OCS/lib/libtcl.a \`
	`$SYBASE/$SYBASE_OCS/lib/libcomn.a \`
	`$SYBASE/$SYBASE_OCS/lib/libintl.a \`
	`-lm -o program`

Linux	`cc -I$SYBASE/$SYBASE_OCS/include`
	`-L$SYBASE/$SYBASE_OCS/lib program.c`
	`-lsrv [-lsybdb|-lct] -lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Table 3-3 shows commands for compiling and linking Server-Library applications using debug libraries:

Table 3-3: Debug link-and-compile commands for Server-Library

Platform	Command

Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/include -L$SYBASE/devlib \`
	`program.c -lsrv [-lsybdb | -lct] -lcs -lcomn -ltcl -lintl \`
	`-lnsl -lm -o program`

Platform	Command
IBM	`xlc_r4 -I$SYBASE/include -L$SYBASE/devlib program.c -lsrv \ [-lsybdb| -lct] -lcs -lcomn -ltcl -lintl -lm - o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/devlib program.c -lsrv \ [-lsybdb | -lct] -lcs -lcomn -ltcl -lintl - linsck \ -Wl,-a,archive -lcl -lm -lBSD -o program`
SGI	`cc [-n32 | -n64] -mips3 -I$SYBASE/include - L$SYBASE/devlib \ program.c -lsrv [-lsybdb | -lct] -lcs -ltcl - lcomn \ -lintl -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include program.c \ -L$SYBASE/$SYBASE_OCS/devlib program.c \ -lsrv [-lsybdb | -lct] -lcs -ltcl \ -lcomn -lintl \ -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib program.c -lsrv [-lsybdb|-lct] -lcs -lsybtcl -lcomn - lintl -rdynamic -ldl -lnsl -lm -o program`

Table 3-4 shows commands for compiling and linking Server-Library applications using shareable libraries (with dynamic drivers):

Table 3-4: Shareable link-and-compile commands for Server-Library

Platform	Command
Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/include - L$SYBASE/lib \ -R$SYBASE/lib program.c -lsrv [-lsybdb | -lct] -lcs -lcomn \ -ltcl -lintl -lnsl -ldl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib program.c - lsrv \ [-lsybdb | -lct] -lcs -ltcl -lcomn -lintl - linsck \ -Wl -lcl -lm -lBSD -o program`
SGI	`cc [-n32 | -n64] -mips3 -I$SYBASE/include - L$SYBASE/lib program.c \ -lsrv [-lsybdb | -lct] -lcs -ltcl -lcomn -lintl -lm -o program`

Platform	Command
HP Tru64 UNIX	`cc -I$SYBASE/$SYBASE_OCS/include program.c \ -L$SYBASE/$SYBASE_OCS/devlib program.c \ -lsrv [-lsybdb | -lct] -lcs -ltcl \ -lcomn -lintl \ -lm -o program`

Note The Open Server program can use Client-Library or DB-Library routines. The bracketed information after -lsrv in the above lines means that you can choose either -lsybdb for DB-Library or -lct for Client-Library.

Kerberos support

Server-Library versions 11.1 and later support Kerberos security features for applications that need a high level of security when communicating over a network. By installing the required Kerberos software and performing the appropriate configuration tasks, your Server-Library applications can take advantage of the following Kerberos security features that are supported in this version:

- Network authentication
- Mutual authentication
- Out-of-sequence authentication
- Replay detection
- Confidentiality

- Integrity

Note With this version of Server-Library, Kerberos support is available on AIX 4.3.3, Linux, Sun Solaris 2.x and HP/UX platforms. Refer to your release bulletin for information regarding additional support.

To develop and run Server-Library applications that take advantage of Kerberos features, perform the tasks listed in Table 3-5:

Table 3-5: Required tasks for Kerberos support

Tasks	For more information
Install the following Kerberos software on your system. Be sure that the *GSS* library support is available as a shared library.	Refer to your Kerberos documentation and to the Open Server *Configuration Guide* for UNIX.
Extract keys for the desired server principal(s) into a key table file using the Kerberos utility called kadmin.	Refer to your Kerberos documentation.
Configure the security section of the *libtcl.cfg* configuration file.	See the Open Client and Open Server *Configuration Guide* for UNIX.
Link your Client-Library application with the Sybase re-entrant libraries.	See “Kerberos support” on page 41.
• For Cybersafe Kerberos: • Set the CSFC5CCNAME environment variable to the credential cache directory location. • Set the CSFC5KTNAME variable to the path of the key table file if other than the default key table file. • For MIT Kerberos • Set the KRB5CCNAME environment variable to the credential cache file location. • Set the KRB5_KTNAME variable to the path of the key table file if other than the default key table file.	Refer to your Kerberos documentation. Default credential cache directory location varies by platform. • For CyberSafe Trustbroker ,the default key table file is */krb5/v5srvtab*. • For MIT Kerberos, the default key table file is */etc/krb5.keytab*.
Use srv_props to set the server principal name if it is different from the server name passed to srv_init.	See the Open Server *Server-Library/C Reference Manual*.

Note To avoid compromising security, Sybase suggests that the *key table* files be owned by the user id that runs Open Server, and that all other users be restricted from accessing this file. Sybase also suggests that each Open Server be run using a unique user id that is not used by interactive processes.

Bulk copy routines

If you plan to use bulk copy routines, link *libblk*, the bulk copy library.

To link in the bulk copy library, add -lblk before -lsrv at the beginning of the link line.

See the Open Client and Open Server *Common Libraries Reference Manual* for more information on bulk copying.

Performance considerations

Linking with shared libraries results in a smaller executable and takes less time than linking with static libraries. However, executables linked with shared libraries may have a slower start-up time than those linked with static libraries. Also, unlike static libraries, the shared libraries must be available at runtime.

The individual requirements of your site determine which type of library will provide the best performance.

Header files

Include the *ospublic.h* header file in all Open Server application source files. Other necessary header files are nested in *ospublic.h*. If Bulk-Library is used, include *bkpublic.h* in addition to *ospublic.h*.

See the Open Server *Server-Library Reference Manual* for more information on header files.

Server-Library sample programs

This section contains information about the sample programs that are included online with Server-Library.

The online sample programs demonstrate typical uses for Server-Library routines in C programs. The sample programs are servers and therefore require entries in the interfaces file or entries in a network directory service to describe their machines and network addresses. See the Open Client and Open Server *Configuration Guide* for UNIX for information on how to configure directory services, including the interfaces file.

Purpose of the sample programs

The sample programs demonstrate specific Open Server functionality. With the exception of ctosdemo, these programs are designed as guides for application programmers, not as Open Server training aids. Read the descriptions at the top of each source file and examine the source code prior to attempting to use the sample programs.

These simplified programs are not intended for use in a production environment. Production-quality programs require additional error-handling and special-case-handling.

Check the individual sample programs to see which trace flags can be used with them. Read the *README* file for complete instructions on running the sample programs.

Note On Sun Solaris, the link line for *secsrv_krb* in the makefile must be updated for the SYBPLATFORM value “dce_sun_svr4” to exclude linking with any Kerberos libraries. The link line should not define the *csfgss_pPtr* symbol. This is defined in the GETKRBLIBS variable at line 169. The line should read:

```
KRBLIBS=" -Bdynamic -lsocket " ;; \
```

Location

The sample programs are located in the *$SYBASE/sample/srvlibrary* directory, which includes:

- Online source code for the sample programs.
- The *makefile* provided to build the samples. Use them as a starting point for your own Server-Library applications.
- The samples header file, *ossample.h*.
- A srv_connect event handler.
- Error handlers.
- The *README* file containing instructions for building, executing, and testing the samples.

Note A client program can stop single and multithreaded samples by executing the stop_srv registered procedure.

Sample program summaries

The following sample programs are included with your software.

Note For the multithread sample, a client uses the stop_serv registered procedure to stop the example.

ctosdemo.c

The *ctosdemo.c* program is an Open Server gateway application that uses Server-Library calls and Client-Library calls. It accepts commands from a client and passes them to a remote Adaptive Server. Then, it retrieves the results from the remote server and passes them to the client. This program *ctosdemo.c* processes a variety of client commands:

- Bulk-copy commands
- Cursor commands
- Dynamic SQL commands
- Language commands
- Option commands
- Remote procedure calls (RPCs)

In addition, it responds to attention requests from a client by calling the srv_attention event handler. It includes an event handler routine to process each type of client command.

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

exfds.c

The *exfds.c* program demonstrates how an Open Server application can service external file descriptors without blocking the entire Open Server process. This program performs a number of tasks:

- Verifies that the current platform supports srv_poll, using the srv_capability routine
- Opens two UNIX pipes
- Spawns two service threads, srv_poll and srv_stop, using the srv_spawn routine

The two service threads implement a simple command/response protocol by writing messages on the UNIX pipes. srv_poll is used to allow Open Server to reschedule the service thread while waiting for a message. Information is written to *srv.log* to monitor the progress. The Open Server performs the command/response protocol the number of times specified in the source code and then queues a SRV_STOP event.

This sample does not require a client application. Check the *srv.log* file for messages to determine if it has started correctly.

fullpass.c

The *fullpass.c* program is an Open Server gateway application that uses the Sybase Tabular Data Stream™ (TDS) passthrough mode. For more information on TDS passthrough, see the “Passthrough Mode” topics page in Chapter 2 of the Open Server *Server-Library/C Reference Manual.*

The event handler routine receives client requests through srv_recvpassthru and forwards this information to an Adaptive Server using the ct_sendpassthru routine. After the entire client command has been forwarded to the remote server, the event handler reads results from the remote server through ct_recvpassthru and returns them to the client using srv_sendpassthru.

The application also includes a SRV_CONNECT event handler. This handler uses srv_getloginfo and ct_setloginfo to forward client connection information to the remote server. It then uses ct_getloginfo and srv_setloginfo to return connection acknowledgment information to the client. All Open Server applications that use TDS passthrough mode must include these calls in their SRV_CONNECT event handler.

intlchar.c

This sample program demonstrates Open Server's handling of national languages and character sets. It initializes values for the Open Server application's national language and character set, and then changes these values in response to client requests.

Client requests come in the form of option commands and language commands. *intlchar.c* installs SRV_OPTION and SRV_LANGUAGE event handlers, as well as a SRV_CONNECT handler.

lang.c

The *lang.c* program demonstrates the use of a srv_language event handler. The event handler responds to client language commands with an informational message, which it sends to the client using the srv_sendinfo routine. This program also contains a srv_connect event handler and error handlers.

For more information on processing language commands, see the "Language Calls" topics page in the Open Server *Server-Library/C Reference Manual*.

multthrd.c

The *multthrd.c* program illustrates a number of Open Server multithreaded programming features, including:

- Creation of a service thread using srv_spawn
- Interthread communication between client connection threads and the service thread through message queues (using srv_getmsgq and srv_putmsgq)
- Sleep and wake-up mechanisms (using srv_sleep and srv_wakeup)
- The use of a callback routine (using srv_callback) to report scheduling information

A service thread logs all the language queries received by this Open Server application.

In the application's language handler, the client thread reads the query from a client and sends a message to the service thread, known as the "logger," with the query as the message data. Then, the client thread waits (srv_sleep). When the service thread gets the message, it wakes up the client thread (srv_wakeup). The logger thus continuously loops, waiting for messages. When it receives a message, it prints the contents of the query to a file and wakes up the sender.

The logger and client threads install SRV_C_RESUME, SRV_C_SUSPEND, SRV_C_TIMESLICE, and SRV_C_EXIT callback handlers to print scheduling information. The *multthrd.c* program installs a SRV_START handler, a SRV_LANGUAGE handler, a SRV_CONNECT handler, and callback handlers.

osintro.c

The *osintro.c* program demonstrates the basic components of an Open Server application. It has no event handlers installed.

regproc.c

The *regproc.c* program demonstrates the use of registered procedures in Open Server versions 11.1 and later. The application registers several procedures at start-up time, and then waits for client commands. No Open Server event handlers are installed.

Clients send RPC commands to execute the registered procedures defined in *regproc.c*.

Several additional client programs are provided for use with *regproc.c*:

- *version.c* – executes a registered procedure (rp_version) that returns the version of the Open Server
- *dbwait.c* – implemented with DB-Library, registers with the Open Server to be notified when the registered procedure rp_version is executed
- *ctwait.c* – implemented with Client-Library, registers with the Open Server to be notified when the registered procedure rp_version is executed

secsrv.c

The *secsrv.c* program demonstrates how Open Server uses network-based security services. The connection handler in this example program retrieves the security properties of the client thread and sends messages to the client that describe which security services are active for the session.

For more information on security services, refer to the Open Client and Open Server *Configuration Guide* for UNIX.

sigalarm.c

The *sigalarm.c* program demonstrates how an Open Server application can use a UNIX SIGALARM signal to schedule periodic events. Specifically, *sigalarm.c*:

- Spawns, using srv_spawn, a service thread that sleeps until an alarm wakes it up. Each time the service thread is awakened, it writes a message to the Open Server log file using the srv_log routine.
- Installs a SIGALARM handler, using the srv_signal routine, that wakes up a sleeping service thread each time the SIGALARM handler is called. *sigalarm.c* requests that a SIGALARM be delivered at a particular interval, using the UNIX alarm call.

This sample does not require a client application. Check the *srv.log* file for messages to determine if it has started correctly.

CHAPTER 4

Open Client Embedded SQL/COBOL

Embedded SQL is a superset of Transact-SQL® that lets you embed Transact-SQL statements in application programs written in a language like COBOL. Embedded SQL includes all Transact-SQL statements, and the extensions needed to use Transact-SQL in an application.

Embedded SQL/COBOL provides a simple way to retrieve, insert, or modify data stored in any Adaptive Server database.

General instructions

To run Embedded SQL/COBOL applications, including the sample programs, you must:

- Be able to access an Adaptive Server on which the pubs2 sample database is installed. Refer to your installation guide for information on installing the pubs2 database.
- Set the following environment variables, which are described in Appendix B, "Environment Variables":
  - SYBASE
  - COBDIR
  - PATH
  - SYBPLATFORM
  - Platform-specific library path variable

Building an Embedded SQL/COBOL executable

This section gives information on libraries, linking, and the header files.

Libraries

The following table lists libraries that you should include if you want to take full advantage of all Embedded SQL/COBOL capabilities. The first row in the table lists libraries that all platforms can use. Subsequent rows list libraries specific for each platform:

Table 4-1: Platform-specific libraries for Embedded SQL/COBOL

Platform	Supported libraries
All platforms	*libcobct* – COBOL interface to Client-Library and CS-Library (Sybase) *libct* – Client-Library (Sybase) *libcs* – CS-Library (Sybase) *libcomn* – An internal shared-utility library (Sybase internal) *libintl* – Internationalization support library (Sybase internal) *libtcl* – Transport Control Layer (Sybase internal)

There are three basic steps to building an executable program from an Embedded SQL/COBOL application:

1 Precompile the application.

2 Compile and link the COBOL source code generated by the precompiler.

3 Load any precompiler-generated stored procedures.

These steps are described in the following sections.

Precompiling the application

The format of the statement to precompile an Embedded SQL/COBOL source program is:

```
cobpre [-a] [-b] [-c] [-d] [-e] [-f]
    [-l] [-m] [-q] [-r] [-v] [-w] [-x] [-y]
   [-Ccompiler]
   [-Ddatabase_name]
   [-Ffips_level]
   [-G[isql_file_name]]
   [-Iinclude_directory]...
   [-Jlocale_for_charset]
```

    [-K*syntax_level*]
    [-L[*listing_file_name*]]
    [-N*interfaces_file_name*]
    [-O*target_file_name*]
    [-P[*password*]]
    [-S*server_name*]
    [-T*tag_id*]
    [-U*user_id*]
    [-V*version_number*]
    [-Z*locale_for_messages*]
    [@*file_name*]
program[.*ext*] [*program*[.*ext*]]...

program is the name of the Embedded SQL/COBOL source file. The default extension for *program* is ".pco." cobpre generates an output file with a ".cbl" extension.

Some of the options are switches that activate features of the precompiler, such as generating stored procedures. These features are "off" by default, and are turned "on" by including the option on the cobpre statement line. Other statement qualifiers specify values for the preprocessor, for example, a password. Enter the value after the option (with or without intervening spaces).

If you enter an invalid option, the precompiler lists the options that are available.

See Appendix A, "Commands and Utilities," for detailed descriptions of the cobpre options.

Compiling and linking the application

The following tables list the general forms of the commands for compiling and linking Embedded SQL/COBOL applications on Sybase-supported platforms running the UNIX operating system.

Table 4-2 shows commands for compiling and linking Embedded SQL/COBOL applications using non-debug libraries.

Table 4-2: Non-debug link-and-compile commands for Embedded SQL/COBOL

Platform	Command
Sun Solaris 2.x	`cob -x program.cbl -L $SYBASE/lib -lcobct -lct -lcs -ltcl \ -lcomn -lintl -ltli -lnsl -lm -o program`
HP 9000(8xx)	`cob -x program.cbl -L $SYBASE/lib -lcobct -lct -lcs -ltcl \ -lcomn -lintl -linsck -lBSD -lm -o program`
IBM RS/6000	`cob -x program.cbl -L $SYBASE/lib -lcobct -lct -lcs -ltcl \ -lcomn -lintl -linsck -lm -o program`
SGI	`cob -x program.cbl -L $SYBASE/lib -lcobct -lct -lcs -ltcl \ -lcomn`
HP Tru64 UNIX	`cobol -ansi -names upper -x program.cbl -L $SYBASE/lib\ -lcobct -lct -lcs -lcomn -ltcl -lintl -lm -o program`

Table 4-3 shows commands for compiling and linking Embedded SQL/COBOL applications using debug libraries.

Table 4-3: Debug link-and-compile commands for Embedded SQL/COBOL

Platform	Command
Sun Solaris 2.x	`cob -g -x program.cbl -L $SYBASE/devlib -lcobct -lct -lcs \ -lcomn -ltcl -lintl -ltli -lnsl -lm -o program`
HP 9000(8xx)	`cob -g -x program.cbl -L $SYBASE/devlib -lcobct -lct -lcs \ -ltcl -lcomn -lintl -linsck -lm -o program`
IBM RS/6000	`cob -g -x program.cbl -L $SYBASE/devlib -lcobct -lct -lcs \ -ltcl -lcomn -lintl -linsck -lm -o program`
HP Tru64 UNIX	`cobol -ansi -names upper -x program.cbl -L$SYBASE/devlib \ -lcobct -lct -lcs -lcomn -ltcl -lintl -lm -o program`

New compile-and-link command lines for HP Tru64 UNIX

Use the following general command form to compile and link an Embedded SQL/COBOL application for DEC COBOL:

Nonthreaded non-reentrant libraries

- Optimized libraries – nonthreaded, non-reentrant:

```
cobol -ansi -names upper -x program.cbl \
 $SYBASE/$SYBASE_OCS/lib/libcobct.a \
 $SYBASE/$SYBASE_OCS/lib/libct.a \
 $SYBASE/$SYBASE_OCS/lib/libcs.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl.a \
 $SYBASE/$SYBASE_OCS/lib/libintl.a \
 -lm -o program
```

- Debug libraries – nonthreaded, non-reentrant:

```
cobol -ansi -names upper -x program.cbl \
 $SYBASE/$SYBASE_OCS/devlib/libcobct.a \
 $SYBASE/$SYBASE_OCS/devlib/libct.a \
 $SYBASE/$SYBASE_OCS/devlib/libcs.a \
 $SYBASE/$SYBASE_OCS/devlib/libcomn.a \
 $SYBASE/$SYBASE_OCS/devlib/libtcl.a \
 $SYBASE/$SYBASE_OCS/devlib/libintl.a \
 -lm -o program
```

Use the following general command form to compile and link an Embedded SQL/COBOL application for MicroFocus COBOL:

- Optimized libraries – nonthreaded, non-reentrant:

```
cob -x program.cbl \
 $SYBASE/$SYBASE_OCS/lib/libcobct.a \
 $SYBASE/$SYBASE_OCS/lib/libct.a \
 $SYBASE/$SYBASE_OCS/lib/libcs.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl.a \
 $SYBASE/$SYBASE_OCS/lib/libintl.a \
 -lm -o program
```

Threaded reentrant libraries

- Optimized libraries – threaded, reentrant:

```
cobol -ansi -names upper -x program.cbl \
 $SYBASE/$SYBASE_OCS/lib/libcobct_r.a \
 $SYBASE/$SYBASE_OCS/lib/libct_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcs_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn_r.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl_r.a \
 $SYBASE/$SYBASE_OCS/lib/libintl_r.a \
 -threads -lm -o program
```

- Debug libraries – threaded, reentrant:

```
cobol -ansi -names upper -x program.cbl \
 $SYBASE/$SYBASE_OCS/devlib/libcobct_r.a \
 $SYBASE/$SYBASE_OCS/devlib/libct_r.a \
 $SYBASE/$SYBASE_OCS/devlib/libcs_r.a \
 $SYBASE/$SYBASE_OCS/devlib/libcomn_r.a \
 $SYBASE/$SYBASE_OCS/devlib/libtcl_r.a \
 $SYBASE/$SYBASE_OCS/devlib/libintl_r.a \
-threads -lm -o program
```

Loading stored procedures

If you use the precompiler -G flag to generate stored procedures, use an isql statement to load the stored procedures into Adaptive Server before you execute the program. The format of the isql statement to execute a generated script is:

```
isql -Uuserid -Ppassword < program.sql
```

where the -U and -P flags specify the user ID and password to log in to Adaptive Server.

See Appendix A, “Commands and Utilities” for a description of isql.

Embedded SQL/COBOL sample programs

The Embedded SQL/COBOL precompiler has two online sample programs, described in the following sections, that demonstrate typical Embedded SQL applications.

Note Before compiling and running the sample programs, copy the contents of *$SYBASE/sample/esqlcob* into a "working" directory, where you can freely experiment with the sample programs without affecting the integrity of the original files.

Purpose of the sample programs

The sample programs demonstrate specific Embedded SQL/COBOL functionality. These programs are designed as guides for application programmers, not as Embedded SQL/COBOL training aids. Read the descriptions at the top of each source file and examine the source code prior to attempting to use the sample programs.

Edit the samples. Before you precompile the programs, replace the user name and password with values that are valid for your Adaptive Server. Comments in the programs show where you should make the changes. Read the *README* file for complete instructions on running the sample programs.

These simplified programs are not intended for use in a production environment. Production-quality programs require additional code to handle errors and special cases.

Location

The sample programs are located in the following directory:

```
$SYBASE/sample/esqlcob
```

This directory includes:

- Online source code for the sample programs.
- The *makefile* provided to build the samples. Use the *makefile* as a starting point for your own Server-Library applications.

- The *README* file containing instructions for building, executing, and testing the samples.

Note Before the sample programs produce results, you may need to press Return.

Example 1: Using cursors for database query

Example 1 shows how to use cursors in an interactive query program. Following is the sample's program flow. The program:

- Displays a list of book types; user selects one type
- Displays all titles in the selected book type; prompts for a title ID
- Displays detailed information about the selected title and continues prompting for title IDs
- Exits the program when Return is pressed at a prompt

Example 2: Displaying and editing rows in a table

Example 2 demonstrates updating a row through a cursor. The program:

- Displays the columns in the authors table row by row.
- Lets the user update author information in all but the au_id column. If the user presses Return for column information, that column's data remains unchanged.
- Requires the user to confirm the update before sending the data to Adaptive Server.

CHAPTER 5 Open Client Embedded SQL/C

Embedded SQL is a superset of Transact-SQL that lets you embed Transact-SQL statements in application programs written in languages like C. Embedded SQL includes all Transact-SQL statements, and the extensions needed to use Transact-SQL in an application program.

Embedded SQL provides a simple way to retrieve, insert, or modify data stored in any Adaptive Server database.

This chapter covers the following topics:

Topic	**Page**
General instructions	59
Building an Embedded SQL/C executable	60
Embedded SQL/C sample programs	67

General instructions

To run Embedded SQL/C applications, including the sample programs, you must:

- Set the following environment variables, which are described in Appendix B, “Environment Variables”:
  - SYBASE
  - SYBPLATFORM
  - Platform-specific library path variable
- Be able to access an Adaptive Server on which the pubs2 sample database is installed. Refer to the Sybase Adaptive Server Enterprise *Installation Guide* for information on installing the pubs2 database.
- Set execute permission on the *sybopts.sh* file for the file’s owner:

```
chmod u+x sybopts.sh
```

- If you have not already done so, include the current directory in the search path:

```
setenv PATH .:$PATH
```

Building an Embedded SQL/C executable

To build an executable program from an Embedded SQL application:

1. Precompile the application.
2. Compile the C source code generated by the precompiler and link your application to any necessary files and libraries.
3. Load any precompiler-generated stored procedures.

The following sections describe these steps.

Precompiling the application

The format of the statement to precompile a source program is:

```
cpre [-a] [-b] [-c] [-d] [-e] [-f]
      [-l] [-m] [-p] [-r] [-s] [-v] [-w] [-x] [-y]
     [-Ccompiler]
     [-Ddatabase_name]
     [-Ffips_level]
     [-G[isql_file_name]]-H
     [-Iinclude_directory]...
     [-Jlocale_for_charset]
     [-Ksyntax_level]
     [-L[listing_file_name]]
     [-Ninterfaces_file_name]
     [-Otarget_file_name]
     [-P[password]]
     [-Sserver_name]
     [-Ttag_id]
     [-Uuser_id]
     [-Vversion_number]
     [-Zlocale_for_messages]
     [@file_name]
 program[.ext] [program[.ext]]...
```

program is the name of the Embedded SQL/C source file. The default extension for *program* is ".cp". cpre generates an output file with a ".c" extension.

Some of the options are switches that activate features of the precompiler. For example, an option can generate a stored procedure. These features are "off" by default, and are turned "on" by including the option on the cpre statement line. Other statement qualifiers specify values for the preprocessor—a password, for example. Enter the value after the option (with or without intervening spaces).

If you enter an invalid option, the precompiler lists the options that are available.

See Appendix A, "Commands and Utilities," for detailed descriptions of precompiler options.

Compiling and linking the application

This section gives information on libraries, linking, and header files.

Note Client-Library and Server-Library now support dynamic loading of Net-Library, directory, and security drivers. This change affects the way you link Client-Library, Server-Library, and Embedded SQL applications.

From this version onward, you no longer need to explicitly link the following object files with your applications:

- Sybase Net-Library drivers (linker option -linsck for HP-UX, HP Tru64 UNIX, SGI, and IBM RS/6000, -ltli for Sun Solaris 2.x)
- Sybase directory or security drivers (linker options -lddce and -lsdce)

The following tables list the general forms of the commands for compiling and linking Embedded SQL/C applications on Sybase supported platforms running the UNIX operating system.

Table 5-1 shows the commands for compiling and linking Embedded SQL/C applications using static libraries.

Table 5-1: Static link-and-compile commands for Embedded SQL/C

Platform	Command
Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/include -L$SYBASE/lib \` *`APP_FILES -lct`* `-lcs -ltcl -lcomn -lintl -Bdynamic -lnsl -ldl -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/include -L$SYBASE/lib` *`APP_FILES -lct`* `\ -lcs -ltcl -lcomn -lintl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib` *`APP_FILES`* `[-W1,-a,archive] \` *`-lct`* `-lcs -ltcl -lcomn -lintl -Wl,-a,default -lcl -lm \ -lBSD -ldld -Wl, -E, +s -o program`
SGI	`cc -o [-n32 | -n64] -mips3 -I$SYBASE/include -Bstatic` *`APP_FILES`* `\ -L$SYBASE/lib` *`-lct`* `-lcs -ltcl -lcomn -lintl \ -Bdynamic -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/include -L$SYBASE/lib` *`APP_FILES -lct`* `-lcs -ltcl \ -lcomn -lintl -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/lib` *`APP_FILES -lct`* `-lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Table 5-2 shows the commands for compiling and linking Embedded SQL/C applications using debug libraries.

Table 5-2: Debug link-and-compile commands for Embedded SQL/C

Platform	Command
Sun Solaris 2.x	`/opt/SUNWspro/bin/cc -I$SYBASE/include -L$SYBASE/devlib \` `-g` *`APP_FILES -lct`* `-lcs -ltcl -lcomn -lintl -Bdynamic` `-lnsl -ldl -lm -o program`
IBM RS/6000	`xlc_r4 -I$SYBASE/include -L$SYBASE/devlib -g` *`APP_FILES`* `\` *`-lct`* `-lcs -ltcl -lcomn -lintl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/devlib -g` *`APP_FILES`* `\` `[-Wl,-a,archive]` *`-lct`* `-lcs -ltcl -lcomn -lintl \` `-Wl,-a,default -lcl -lm -lBSD -ldld -Wl, -E, +s -o program`
SGI	`cc -g [-n32 | -n64] -mips3 -I$SYBASE/include -L$SYBASE/devlib \` *`APP_FILES -lct`* `-lcs -ltcl -lcomn -lintl -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/include -L$SYBASE/devlib` *`APP_FILES -lct`* `-lcs \` `-ltcl _oldstyle_liblookup -lcomn -lintl -lnsl -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include` `-L$SYBASE/$SYBASE_OCS/devlib` *`APP_FILES`* *`-lct`* `-lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Table 5-3 shows the commands for compiling and linking Embedded SQL/C applications using shareable libraries (with dynamic drivers).

Table 5-3: Shareable link-and-compile commands for Embedded SQL/C

Platform	Command
Sun Solaris 2.x	`cc -I$SYBASE/include -L$SYBASE/lib APP_FILES \ $SYBASE/include/sybesql.c -lct -lcs -ltcl \ -lcomn -lintl -ltli -lnsl -ldl -lm -o program`
HP 9000(8xx)	`cc -I$SYBASE/include -L$SYBASE/lib APP_FILES \ $SYBASE/include/sybesql.c -lct -lcs -ltcl \ -lcomn -lintl -linsck -Wl -lcl -lm -lBSD -o program`
SGI	`cc [-n32 | -n64] -mips3 -I$SYBASE/include -L$SYBASE/lib APP_FILES \ -lct -lcs -ltcl -lcomn -lintl -ldl -lm -o program`
HP Tru64 UNIX	`cc -I$SYBASE/include -L$SYBASE/lib APP_FILES \ -oldstyle_liblookup -lct -lcs -ltcl-lcomn \ -lintl -lm -o program`
Linux	`cc -I$SYBASE/$SYBASE_OCS/include -L$SYBASE/$SYBASE_OCS/devlib APP_FILES -lct -lcs -lsybtcl -lcomn -lintl -rdynamic -ldl -lnsl -lm -o program`

Note The object produced by compiling the *sybesql.c* file contains utility routines that are used by Embedded SQL/C applications. You must link *sybesql.o* in with every application for the application to work properly.

- The link line for an Embedded SQL/C application is identical to that used for a Client-Library application. In the link line, APP_FILES should include the following information, in the order given below:
  - Any generated C files (or *.o* files compiled from generated C files).
  - *$SYBASE/include/sybesql.o* (Make sure, if you upgrade, that you are using the latest version of this file.)
- *-lct* represents the linker options to link in the Open Client and Open Server libraries that your code calls. These options can be specified by any or all of the following linker options, in the order shown:

For non-DCE applications	For DCE applications
-lsrv (for Server-Library routines)	-lsrv_r (for Server-Library routines)
-lblk (for Bulk-Library routines)	-lblk_r (for Bulk-Library routines)
-lct (for Client-Library routines)	-lct_r (for Client-Library routines)

For HP-UX system users:

- The option -W1,-a,archive causes the linker to statically link the Sybase libraries. Without it, shared versions of the Sybase libraries are used. In this case, the SH_LIB_PATH environment variable must include *$SYBASE/lib* at runtime, and the application user must have read and execute permission on the libraries in *$SYBASE/lib*.
- HP-UX will not use the SH_LIB_PATH environment variable at runtime unless the application is linked with the +s linker option. You must use the +s linker options so that the system will be able to find Sybase libraries at runtime. -E is required to prevent undefined-symbol errors when driver libraries are loaded at runtime. See the HP-UX ld man page for more information.

New compile-and-link lines for HP Tru64 UNIX

Use the following general command form to compile and link an Embedded SQL/C application:

Nonthreaded non-reentrant libraries

- Optimized libraries – nonthreaded, non-reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include program.c \
 $SYBASE/$SYBASE_OCS/lib/libct.a \
 $SYBASE/$SYBASE_OCS/lib/libcs.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn.a \
 $SYBASE/$SYBASE_OCS/lib/libintl.a \
 -lm -o program
```

- Debug libraries – nonthreaded, non-reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include
-L$SYBASE/$SYBASE_OCS/devlib program.c \
 -lct -lcs -ltcl -oldstyle_liblookup \
 -lcomn -lintl -lnsl \
 -lm -o program
```

- Shareable libraries – nonthreaded, non-reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include
-L$SYBASE/$SYBASE_OCS/lib program.c \
 -oldstyle_liblookup -lct -lcs \
 -ltcl -lcomn -lintl \
 -lm -o program
```

Threaded reentrant libraries

- Optimized libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include program.c -threads
```

```
\
 $SYBASE/$SYBASE_OCS/lib/libct_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcs_r.a \
 $SYBASE/$SYBASE_OCS/lib/libtcl_r.a \
 $SYBASE/$SYBASE_OCS/lib/libcomn_r.a \
 $SYBASE/$SYBASE_OCS/lib/libintl_r.a \
 -lm -o program
```

- Debug libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include
-L$SYBASE/$SYBASE_OCS/devlib -g \
 -threads program.c \
 -lct_r -lcs_r -ltcl_r -lcomn_r-lintl_r \
 -oldstyle_liblookup \
 -lm -o program
```

- Shareable libraries – threaded, reentrant:

```
cc -I$SYBASE/$SYBASE_OCS/include
-L$SYBASE/$SYBASE_OCS/lib program.c \
 -threads -oldstyle_liblookup \
 $SYBASE/$SYBASE_OCS/include/sybesql.c \
 -lct_r -lcs_r -ltcl_r \
 -lcomn_r
 -lintl_r -lnsl_r \
 -lm -o program
```

Performance considerations

Linking with shared libraries results in a smaller executable and takes less time than linking with static libraries. However, executables linked with shared libraries may have a slower start-up time than those linked with static libraries. Also, unlike static libraries, the shared libraries must be available at runtime.

The type of library that provides the best performance depends on the individual requirements of your site.

Loading stored procedures

If you use the precompiler -G flag to generate stored procedures, use an isql statement to load the stored procedures into Adaptive Server before you execute the program. The format of the isql statement to execute a generated script is:

```
isql -Uuserid -Ppassword < program.sql
```

where the -U and -P flags specify the user ID and password to log in to Adaptive Server.

See Appendix A, "Commands and Utilities," for a description of isql.

Embedded SQL/C sample programs

The Embedded SQL/C precompiler has two online sample programs that demonstrate typical Embedded SQL/C applications.

Purpose of the sample programs

The sample programs demonstrate specific Embedded SQL/C functions. These programs are designed as guides for application programmers, not as Embedded SQL/C training aids. Read the descriptions at the top of each source file and examine the source code prior to attempting to use the sample programs.

These simplified programs are not intended for use in a production environment. Production-quality programs require additional error handling. Read the *README* file for complete instructions on running the sample programs.

Location

The sample programs are located in the directory:

```
$SYBASE/sample/esqlc
```

This directory includes:

- Online source code for the sample programs.
- The *makefile* provided to build the samples. Use the *makefile* as a starting point for your own Embedded SQL applications.
- The samples header file, *sybsqlex.h*.
- The *README* file containing instructions for building, executing, and testing the samples.

Before compiling and running the sample programs, copy the contents of *$SYBASE/sample/esql* into a “working” directory, where you can freely experiment with the sample programs without affecting the integrity of the original files.

Header file

Before you precompile the programs, you must edit the sample header file, described below, and replace the user name and password with values that are valid for your Adaptive Server. Comments in the programs show where you should make the changes.

All of the sample programs reference the sample header file, *sybsqlex.h*. The contents of *sybsqlex.h* are as follows:

```
/**************************************************
 *                                                *
 *  sybsqlex.h - header file for Embedded SQL/C   *
 *examples                                        *
 *                                                *
 **************************************************/

 #define USER             "user name"
 #define PASSWORD          "password"
```

All of the samples contain this line:

```
#include "sybsqlex.h"
```

USER and PASSWORD are defined in *sybsqlex.h* as user name and password. Before running the sample programs, you must edit *sybsqlex.h* and change user name to your Adaptive Server login name and “password” to your Adaptive Server password.

Example 1: Using cursors for database query

The *example1.cp* program shows how to use cursors in an interactive query program. The program:

- Displays a list of book types; user selects one type
- Displays all titles in the selected book type; prompts for a title ID
- Displays detailed information about the selected title and continues prompting for title IDs

- Exits when Return is pressed at a prompt

Example 2: Displaying and editing rows of a table

example2.cp demonstrates updating a row through a cursor. The program:

- Displays the columns in the authors table, row by row.
- Lets the user update author information in all but the au_id column. If the user presses Return for column information, that column's data remains unchanged.
- Requires the user to confirm the update before sending the data to Adaptive Server.

APPENDIX A

Commands and Utilities

This appendix contains reference pages for the following utilities:

- bcp
    - bcp – copies a database table to or from an operating system file in a user-specified format.
- cobpre; cpre
    - cobpre – precompiles a COBOL source program to produce target, listing, and isql files.
    - cpre – precompiles a C source program to produce target, listing, and isql files.
- defncopy
    - defncopy – copies definitions for specified views, rules, defaults, triggers, procedures, or reports from a database to an operating system file or from an operating system file to a database.
- isql
    - isql – interactively parses SQL to Adaptive Server.

With this version, the messages generated by the bcp, defncopy, and isql utilities have changed. If you process these messages with scripts that parse specific strings (such as with awk or grep), you may need to change the search patterns of these scripts to accommodate the new messages.

bcp

Description

Copies a database table to or from an operating system file in a user-specified format.

Syntax

```
bcp [[database_name.]owner.]table_name
    {in | out} datafile
 [-c] [-E] [-n] [-N] [-v] [-X]
```

[-a *display_charset*]
[-A *size*]
[-b *batchsize*]
[-e *errfile*]
[-f *formatfile*]
[-F *firstrow*]
[-I *interfaces_file*]
[-J *client_charset*]
[-K *keytab_file*]
[-L *lastrow*]
[-m *maxerrors*]
[-q *datafile_charset*]
[-r *row_terminator*]
[-R *remote_server_principal*]
[-S *server*]
[-t *field_terminator*]
[-T *text_or_image_size*]
[-U *username*]
[-V [*security_options*]
[-Y]
[-z[*language*]
[-Z *security_mechanism*]

Parameters

database_name
is optional if the table being copied is in your default database or in *master*. Otherwise, you must specify a database name.

owner
is optional if you or the Database Owner own the table being copied. If you do not specify an owner, bcp looks first for a table of that name owned by you, then it looks for one owned by the Database Owner. If another user owns the table, you must specify the owner name or the command fails.

table_name
is the name of the database table or view to copy.

in | out
is the direction of the copy. in indicates a copy from a file into the database table, and out indicates a copy to a file from the database table.

datafile
is the name of an operating system file.

-a *display_charset*
allows you to run bcp from a terminal where the character set differs from that of the machine on which bcp is running. -a in conjunction with -J specifies the character set translation file (*.xlt* file) required for the conversion. Use -a without -J only if the client character set is the same as the default character set.

-A *size*

specifies the network packet size to use for this bcp session. For example:

```
bcp -A 2048
```

sets the packet size to 2048 bytes for this bcp session. *size* must be between the values of the default network packet size and maximum network packet size configuration variables, and it must be a multiple of 512.

Use larger-than-default network packet sizes to improve the performance of large bulk-copy operations.

-b *batchsize*

is the number of rows per batch of data copied (the default is to copy all the rows in one batch). Batching applies only when bulk copying in; it has no effect on bulk copying out.

-c

performs the copy operation with char as the storage type of all columns in the data file. This option does not prompt for each field; it uses *char* as the storage type, no prefixes, \t (tab) as the default field terminator, and \n (newline) as the default row terminator.

-e *errfile*

is the name of an error file where bcp stores any rows that it was unable to transfer from the file to the database. Error messages from the bcp program appear on your terminal. bcp creates an error file only when you specify this option.

-E

explicitly specifies the value of a table's IDENTITY column.

By default, when you bulk copy data into a table with an IDENTITY column, bcp assigns each row a temporary IDENTITY column value of 0. This is only effective when copying data into a table. bcp reads the value of the ID column from the data file, but ignores it and does not send it to the server; instead, as bcp inserts each row into the table, the server assigns the row a unique, sequential IDENTITY column value, beginning with the value 1. If you specify the -E flag when copying data into a table, bcp will read the value from the data file and send it to the server, which will insert these values into the table. If the number of inserted rows exceeds the maximum possible IDENTITY column value, Adaptive Server returns an error.

The -E option has no effect when copying data out, in other words, the ID column is copied to the data file (unless the -N option is used).

-f *formatfile*

is the full path name of a file with stored responses from a previous use of bcp on the same table. After you answer bcp's format questions, it prompts you to save your answers in a format file; creation of the format file is optional. The default file name is *bcp.fmt*. The bcp program can refer to a format file when copying data, so that you do not have to duplicate your previous format responses interactively. Use this option only when you previously created a format file that you want to use now for a copy in or out. If this option is not used, bcp queries you for format information interactively.

-F *firstrow*

is the number of the first row to copy (default is the first row).

-I *interfaces_file*

specifies the name and location of the interfaces file to search when connecting to Adaptive Server. If you do not specify -I, bcp looks for an interfaces file located in the directory specified by the SYBASE environment variable.

-J *client_charset*

specifies the character set to use on the client. bcp uses a filter to convert input between *client_charset* and the Adaptive Server character set.

-J *client_charset* requests that Adaptive Server convert to and from *client_charset*, the character set used on the client.

-J with no argument sets character-set conversion to NULL. No conversion takes place. Use this if the client and server use the same character set.

Omitting -J sets the character set to a default for the platform which may not necessarily be the character set that the client is using. Table A-1 lists platform defaults.

Table A-1: Default character sets for different platforms

Platform	Default character set
Sun, Digital, Pyramid, RS6000/AIX, others	iso_1
HP	roman8

-K *keytab_file*

can be used only with DCE security. It specifies a DCE keytab file that contains the security key for the user name specified with -U option. Keytab files can be created with the DCE dcecp utility. See your DCE documentation for more information.

If the -K option is not supplied, the bcp user must be logged in to DCE with the same user name as specified with the -U option.

-L *lastrow*

is the number of the last row to copy (default is the last row).

-m *maxerrors*

is the maximum number of errors permitted before bcp aborts the copy. bcp throws out each row that it cannot build, counting it as one error. If you do not include this option, bcp uses a default value of 10.

-n

performs the copy operation using native (operating system) formats. This option does not prompt for each field. Files in native data format are usually not human-readable.

Warning! Do not use bcp in native format to recover data, salvage data, or resolve an emergency situation. Do not use bcp in native format to transport data between different hardware platforms, different operating systems, or different major versions of Adaptive Server. Using bcp in native format can create flat files that cannot be reloaded into Adaptive Server, and it may be impossible to recover the data. If you are unable to rerun bcp in character format (for example, table truncated/dropped, hardware damage, database dropped), the data will be unrecoverable.

-N

skips the IDENTITY column. Use this option when copying data in if your host data file does not include a place holder for the IDENTITY column values, or when copying data out and you do not want to include the IDENTITY column information in the host file.

You cannot use both -N and -E options when copying in data.

-P *password*
specifies an Adaptive Server password. This option is ignored if -V is used.

-q *datafile_charset*
allows you to run bcp to copy character data to or from a file system that uses a character set different from the client character set. -q in conjunction with -J specifies the character set translation file (*.xlt* file) required for the conversion.

In Japanese language environments, the -q flag translates Hankaku Katakana (half-width characters) into Zenkaku Katakana (full-width characters). Use with the argument "*zenkaku*" and with the -J flag to indicate the client's Japanese character set (sjis or eucjis). The *zenkaku.xlt* file was designed to translate *only* from terminal display to Adaptive Server, *not* from Adaptive Server to the terminal.

Note The ascii_7 character set is compatible with all character sets. If either Adaptive Server's or the client's character set is set to ascii_7, any 7-bit ASCII character is allowed to pass between client and server unaltered. Other characters produce conversion errors. Character set conversion issues are covered more thoroughly in the *System Administration Guide.*

-r *row_terminator*
specifies the default row terminator.

-R *remote_server_principal*
specifies the principal name for the server. By default, a server's principal name matches the server's network name (which is specified with the -S option or the DSQUERY environment variable). The -R option must be used when the server's principal name and network name are not the same.

-S *server*
specifies the name of the Adaptive Server to connect to. If you specify -S with no argument, bcp uses the server that your DSQUERY environment value specifies.

-t *field_terminator*
specifies the default field terminator.

-T *text_or_image_size*
allows you to specify, in bytes, the maximum length of text or image data that Adaptive Server sends. The default is 32K. If a text or image field is larger than the value of -T or the default, bcp does not send the overflow.

-U *username*

specifies an Adaptive Server login name. If you do not specify *username*, bcp uses the current user’s operating system login name.

-V *security_options*

specifies network-based user authentication. With this option, the user must log in to the network’s security system before running the utility. In this case, users must supply their network user name with the -U option; any password supplied with the -P option is ignored.

-V can be followed by a *security_options* string of key-letter options to enable additional security services. These key letters are:

c – Enable data confidentiality service

i – Enable data integrity service

m – Enable mutual authentication for connection establishment

o – Enable data origin stamping service

r – Enable data replay detection

q – Enable out-of-sequence detection

-v

reports the current version and copyright message of the bcp program.

-X

specifies that, in this connection to the server, the application initiate the login with client-side password encryption. bcp (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which bcp uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

-Y

specifies that the character-set conversion is disabled in the server, and is performed by bcp on the client side when using bcp IN.

Note All character-set conversion is done in the server during bcp OUT.

-z *language*

is the official name of an alternate language that the server uses to display bcp prompts and messages. Without the -z flag, bcp uses the server’s default language. Add languages to an Adaptive Server at installation, or afterwards with the utility langinstall or the stored procedure sp_addlanguage.

-Z *security_mechanism*
specifies the name of a security mechanism to use on the connection.

Security mechanism names are defined in the *$SYBASE/install/libtcl.cfg* configuration file. If no *security_mechanism* name is supplied, the default mechanism is used. For more information on security mechanism names, see the description of the *libtcl.cfg* file in the Open Client and Open Server *Configuration Guide* for UNIX.

Examples

1. In the following example, the -c option copies data out of the *publishers* table in character format (using *char* for all fields). The -t *field_terminator* option ends each field with a comma, and the -r *row_terminator* option ends each line with a Return. bcp prompts only for a password. The first backslash before the final "r" escapes the second so that one backslash prints.

```
bcp pubs2..publishers out pub_out -c -t , -r \\r
```

2. In the following example, bcp copies data from the publishers table to a file named *pub_out* for later reloading into Adaptive Server. Pressing Return accepts the defaults that the prompts specify. The same prompts appear when copying data into the publishers table.

```
bcp pubs2..publishers out pub_out
 Password:

 Enter the file storage type of field pub_id [char]:
 Enter prefix length of field pub_id [0]:
 Enter length of field pub_id [4]:
 Enter field terminator [none]:

Enter the file storage type of field pub_name [char]:
 Enter prefix length of field pub_name [1]:
 Enter field terminator [none]:

 Enter the file storage type of field city [char]:
 Enter prefix length of field city [1]:
 Enter field terminator [none]:

 Enter the file storage type of field state [char]:
 Enter prefix length of field state [1]:
 Enter field terminator [none]:

 Do you want to save this format information in a
file? [Y] y
 Host filename [bcp.fmt]: pub_form

 Starting copy...
 3 rows copied.
 Clock Time (ms.): total = 300   Avg = 1 (300.00 rows
 per sec.)
```

3 To copy this data back into Adaptive Server using the saved format file, *pub_form*, use the following command:

```
bcp pubs2..publishers in pub_out -f pub_form
```

4 To see a list of possible datatypes, enter "?" at the prompt:

```
Enter the file storage type of field 'pub_id'
 ['char']:?
 Invalid column type. Valid types are:
 <cr>: same type as Adaptive Server column.
     c : char
     T : text
     i : int
     s : smallint
     t : tinyint
     f : float
     m : money
     b : bit
     d : datetime
     x : binary
     I : image
     D : smalldatetime
     r : real
     M : smallmoney
     n : numeric
     e : decimal
```

Enter the single letter exactly as it appears above.

5 The following example copies a data file created with a character set used on a VT200 terminal into the pubs2.publishers table. The -q flag translates it. The -z flag displays bcp messages in French.

```
bcp pubs2..publishers in vt200_data -J iso_1 -q
vt200 -z french
```

6 The following example specifies that Adaptive Server send 40K of *text* or *image* data using a packet size of 4096:

```
bcp publishers out -T 40960 -A 4096
```

Usage

New features

Note If there is an external Sybase configuration file, add this section to enable bcp:

[BCP]

---

bcp for System 11 is built with Client-Library.

The bcp user interface is unchanged except for the following:

- New command-line options have been added to enable network-based security services on the connection as follows:

  -K *keytab_file*
  -R *remote_server_principal*
  -V *security_options*
  -Z *security_mechanism*

- The -y *sybase_directory* option is ignored.
- Error message format is different than previous versions of bcp. If you have scripts that perform routines based on the values of these messages you may need to re-write them, for example:

  The display message that indicates the number of rows transferred has been changed. During a session, this version of bcp periodically reports a running total of rows transferred. This message replaces the "1000 rows transferred" message displayed by the previous bcp.

Note To use a previous version of bcp, you must set the CS_BEHAVIOR property in the [bcp] section of the *ocs.cfg* file:

[bcp]

CS_BEHAVIOR = CS_BEHAVIOR_100

If CS_BEHAVIOR is not set to CS_BEHAVIOR_100, you can use functionality for bcp 11.1 and later.

- bcp provides a convenient, high-speed method for transferring data between a database table or view and an operating system file. It is capable of reading or writing files in a wide variety of formats. When copying in from a file, bcp inserts data to an existing database table; when copying out to a file, bcp overwrites any previous contents of the file.

- Upon completion, bcp informs you of the number of rows of data successfully copied, the total time the copy took, the average amount of time in milliseconds that it took to copy one row, and the number of rows copied per second.
- The bcp utility does not insert any row that contains an entry exceeding the character length of the corresponding target table column. For example, bcp does not insert a row with a field of 300 bytes into a table with a character column length of 256 bytes. Instead, bcp reports a conversion error, for example:

  ```
  cs_convert: cslib user api layer: common library
  error: The result is truncated because the
  conversion/operation resulted in overflow
  ```

  and skips the row. bcp does *not* insert truncated data into the table.

  To keep track of data that violate length requirements, run bcp with the -e *log-file name* option. bcp records the row and column number of the rejected data, the error message, and the data in the log file you specify.

Copying tables with indexes or triggers

- The bcp program is optimized to load data into tables that do not have indexes or triggers associated with them. It loads data into tables without indexes or triggers at the fastest possible speed, with a minimum of logging. Page allocations are logged, but the insertion of rows is not.

  When you copy data into a table that has one or more indexes or triggers, a slower version of bcp is automatically used, which logs row inserts. This includes indexes implicitly created using the unique integrity constraint of a create table statement. However, bcp does not enforce the other integrity constraints defined for a table.

Because the fast version of bcp inserts data without logging it, the System Administrator or Database Owner must first set the system procedure sp_dboption, "DB", true. If the option is not true, and you try to copy data into a table that has no indexes or triggers, Adaptive Server generates an error message. You do not need to set this option to copy data out to a file, or to copy data into a table that contains indexes or triggers.

Note Because bcp logs inserts into a table that has indexes or triggers, the log can grow very large. You can truncate the log with dump transaction after the bulk copy completes, after you have backed up your database with dump database.

- While the select into/bulkcopy option is "on," you cannot dump the transaction log. Issuing dump transaction produces an error message instructing you to use dump database instead.

Warning! Be certain that you dump your database before you turn off the select into/bulkcopy flag. If you insert unlogged data into your database, and then perform a dump transaction before you perform a dump database, you will not be able to recover your data.

- Unlogged bcp runs more slowly while a dump database is taking place.
- Table A-2 shows which version bcp uses when copying in, the necessary settings for the select into/bulkcopy option, and whether the transaction log is kept and if it is dumpable.

Table A-2: Fast bcp vs. slow bcp

	select into/bulk copy	
bcp version	**on**	**off**
Fast bcp	OK	bcp
(no indexes or triggers on target table)	dump transaction prohibited	dump transaction prohibited
Slow bcp	OK	OK
(one or more indexes or triggers)	dump transaction prohibited	dump transaction OK

- By default, the select into/bulkcopy option is “off” in newly created databases. To change the default situation, turn this option “on” in the model database.

Note The performance penalty for copying data into a table that has indexes or triggers in place can be severe. If you are copying in a very large number of rows, it may be faster to drop all the indexes and triggers first with drop index (or alter table for indexes created as a unique constraint) and drop trigger; set the database option; copy the data into the table; re-create the indexes and triggers; and then dump the database. Remember to allocate disk space for the construction of indexes and triggers for a clustered index, about 1.2 times the amount of space needed for the data, in addition to the space needed for the data.

Responding to *bcp* prompts

When you copy data in or out using the -n (native format) or -c (character format) option, bcp only prompts you for your password, unless you supplied it with the -P option. If you do not supply either the -n, -c or -f *formatfile* option, bcp prompts you for information for each field in the table.

- Each prompt displays a default value, in brackets, which you can accept by pressing Return. The prompts include:
  - The file storage type, which can be character or any valid Adaptive Server datatype
  - The prefix length, which is an integer indicating the length in bytes of the following data
  - The storage length of the data in the file for nonNULL fields
  - The field terminator, which can be any character string
  - Scale and precision for numeric and decimal dat types

  The row terminator is the field terminator of the last field in the table or file.
- The bracketed defaults represent reasonable values for the datatypes of the field in question. For the most efficient use of space when copying out to a file:
  - Use the default prompts.
  - Copy all data in their table datatypes.
  - Use prefixes as indicated.

- Do not use terminator.s
- Accept the default lengths.

The following table shows the defaults and possible alternate responses:

Table A-3: bcp prompts, defaults, and responses

Prompt	Default provided	Possible responses
File storage type	Use database storage type for most fields except: char for varchar binary for varbinary	char to create or read a human-readable file; any CS datatype where implicit conversion is supported.
Prefix length	0 for fields defined with char datatype (not storage type) and all fixed-length datatypes. 1 for most other datatypes. 2 for binary and varbinary saved as char. 4 for text and image.	0 if no prefix is desired; defaults are recommended in all other cases.
Storage length	For char and varchar, use defined length. For binary and varbinary saved as char, use default. For all other datatypes, use maximum length needed to avoid truncation or data overflow.	Default values, or greater, are recommended.
Field or row terminator	None.	Up to 30 characters, or one of the following: \t tab \n newline \r carriage return \0 null terminator \ backslash

- bcp can copy data out to a file either as its native (database) datatype, or as any datatype for which implicit conversion is supported for the datatype in question. bcp copies user-defined datatypes as their base datatype or as any datatype for which implicit conversion is supported. For more information on datatype conversions, see cs_convert information in the Open Client-Client Library/C Reference Manual.

  Note Be careful copying data in native format from different versions of Adaptive Server because they do not always have the same datatypes.

- A prefix length is a 1-, 2-, or 4-byte integer that represents the length of each data value in bytes. It immediately precedes the data value in the host file.
- Fields defined in the database as char, nchar, and binary are always padded with spaces (null bytes for binary) to the full length defined in the database. timestamp data is treated as binary(8).

  If data in varchar and varbinary fields is longer than the length you specify for copy out, bcp truncates the data in the file at the specified length.
- A field terminator string can be up to 30 characters long; the most common terminators are a tab (entered as "\t" and used for all columns except the last one), and a new line (entered as "\n" and used for the last field in a row). Other terminators are: "\0" (the null terminator), "\" (backslash), and "\r" (Return). When choosing a terminator, be sure that its pattern does not appear in any of your character data. For example, if you use tab terminators with a string that contains a tab, bcp could not identify which tab represents the end of the string. Because bcp always looks for the first possible terminator, in this case it would find the wrong one.

  When a terminator or prefix is present, it affects the actual length of data transferred. If the length of an entry being copied out to a file is less than the storage length, it is followed immediately by the terminator, or the prefix for the next field. The entry is not padded to the full storage length (char, nchar, and binary data is returned from Adaptive Server already padded to the full length).

  When copying in from a file, data is transferred until either the number of bytes indicated in the "Length" prompt has been copied or the terminator is encountered. Once a number of bytes equal to the specified length has been transferred, the rest of the data is flushed until the terminator is encountered. When no terminator is used, the table storage length is strictly observed.

- The following tables show the interaction of prefix lengths, terminators, and field length on the information in the file. "P" indicates the prefix in the stored table. "T" indicates the terminator, and dashes (- -) show appended spaces. An ellipsis (...) indicates that the pattern repeats for each field. The field length is 8 for each column, and "string" represents the 6-character field each occurence.

Table A-4: Adaptive Server char data

	Prefix length = 0	**Prefix length 1,2,or 4**
No Terminator	string--string--.	Pstring--Pstring--.
Terminator	string--Tstring--T.	Pstring--TPstring--T.

Table A-5: Other datatypes converted to char storage

	Prefix length = 0	**Prefix length 1,2,or 4**
No Terminator	string--string--.	PstringPstring.
Terminator	stringTstringT.	PstringTPstringT.

- Note that the file storage type and length of a column do not have to be the same as the type and length of the column in the database table. (If types and formats copied in are incompatible with the structure of the database table, the copy fails.
- File storage length generally indicates the maximum amount of data to be transferred for the column, excluding terminators and prefixes.
- When copying data into a table, bcp observes any defaults defined for columns and user-defined datatypes. However, bcp ignores rules to load data at the fastest possible speed.
- Because bcp considers any data column that can contain null values to be variable length, use either a length prefix or terminator to denote the length of each row of data.
- Data written to a host file in its native format preserves all of its precision. datetime and float values preserve all of their precision even when they are converted to character format. Adaptive Server stores *money* values to a precision of one ten-thousandth of a monetary unit. However, when money values are converted to character format, their character format values are recorded only to the nearest two places.

- Before copying data that is in character format from a file into a database table, check the datatype entry rules in the "Datatypes" section of the Adaptive Server *Reference Manual*. Character data that is being copied into the database with bcp must conform to those rules. Note especially that dates in the undelimited *(yy)yymmdd* format may result in overflow errors if the year is not specified first.
- When you send host data files to sites that use terminals different from your own, inform them of the *datafile_charset* that you used to create the files.

Messages

```
Error in attempting to load a view of translation
tables.
```

The character translation file(s) named with the -a or -q parameter is missing, or you mistyped the name(s).

cobpre; cpre

Description

cpre precompiles a C source program to produce target, listing, and isql files.

cobpre precompiles a COBOL source program to produce target, listing, and isql files.

Syntax

cobpre|cpre [-a] [-b] [-c] [-d]
    [-e] [-f] [-l] [-m] [-p (cpre)] [-q (cobbre)]
    [-r][-s (cpre)] [-t (cobpre)] [-v] [-w] [-x] [-y]
    [-Ccompiler]
    [-D*database*_name]
    [-F*fips_level*]
    [-G[*isql_file_name*]]
    [-H[*server_name*]]
    [-I*include_directory*]...
    [-J*locale_for_charset*]
    [-K*syntax_level*]
    [-L[*listing_file_name*]]
    [-N*interfaces_file_name*]
    [-O*target_file_name*]
    [-P[*password*]]
    [-S*server_name*]
    [-*Ttag_id*]
    [-Vversion_*number*]
    -Z*locale_for_messages*]

[@*file_name*]
program[.*ext*] [*program*[.*ext*]]... Parameters

Parameters

-a

allows cursors to remain open across transactions. (See the *Sybase Adaptive Server Reference Manual* for information about cursors and transactions. If you do not use this option, cursors behave as though set close on endtran were in effect. This behavior is ANSI-compatible.

-b

disables rebinding of host variable addresses typically used in fetch statements. If you do not use this option, a rebind occurs on every fetch statement unless you specify otherwise in your Embedded SQL/C program.

The -b option differs in the 11.1 and 10.0.x versions of the Embedded SQL precompilers:

- For the 11.1 versions of cpre, the norebind attribute applies to all fetch statements of a cursor whose declaration was precompiled with the -b option.
- For the 10.0.x versions of cpre, the norebind attribute applied to all fetch statements in each Embedded SQL source file precompiled with -b, regardless of where the cursors were declared.

-c

turns on the Client-Library debugging feature. When the -c option is used, Client-Library generates calls to ct_debug. This option is useful during application development but should be turned off for final application delivery.

-d

indicates that delimited identifiers will not be used. The application can send character data to the server in double quotes (“ “).

-e

when processing an exec sql connect statement, directs Client Library to use the external configuration file to configure the connection. Without this option, the precompiler generates Client Library function calls to configure the connection. Refer to the Open Client Client-Library/C *Reference Manual* for information about the external configuration file and the CS_CONFIG_BY_SERVERNAME property.

-f

checks that the SQL contained in the ESQL source file is ANSI SQL-89 compliant. If the user provides a SQL Server name and valid user name/password combination, the precompiler will connect to the server and pass all static SQL to the server for standards compliance checking. -f generates code enabling the server to continue checking the SQL for compliance while an application is being run.

-l

turns off generation of #line directives.

-m

runs the application in Sybase auto-commit mode. Auto-commit mode means that transactions are not chained. Explicit begin and end transactions are required or every statement is immediately committed. If you do not specify this option, the application runs in ANSI-style chained transaction mode.

-p

generates a separate command handle for each SQL statement in the module that has input host variables and enables persistent binds on each command handle. This option improves performance of repeatedly executed commands with input parameters but at the cost of increased storage space and slightly slower response on the first iteration of each such command.

Applications that rely on inserting empty strings instead of NULL strings when the host string variable is empty do not work if the -p option is turned on. The persistent bind implementation prevents Embedded SQL from circumventing Client-Library protocol (which inserts NULL strings).

-q

generates code with double quotes (") rather than a single quote(’).

-r

disables repeatable reads. If you do not use this option, a set transaction isolation level 3 statement is generated and will be executed during connect statements.

-s

includes static function declarations in the precompiled file.

-t

specifies that the input source file is in terminal-mode format (HP Tru64 UNIX).

-v

displays the precompiler version information only (without precompiling).

-w

disables display of precompiler warning and informational messages to the screen or a listing file.

-x

instructs Client-Library to use external configuration files. See the CS_EXTERNAL_CONFIG property described in the Open Client *Client-Library Reference Manual* and the INITIALIZE_APPLICATION statement described in the Embedded SQL *Reference Manual*.

-y

supports CS_TEXT and CS_IMAGE datatypes so they can be used as input host variables. At runtime, the data is directly included into the character string sent to the server. Only static SQL statements are supported; use of text and image as input parameters to dynamic SQL is not supported. This substitution of arguments into command strings is only performed if the -y command line option is used.

-C*compiler*

specifies the target host language compiler.

cpre – Possible values are “ansi_c” for ANSI standard C and “kr_c” for Kernighan and Ritchie C. If you do not use this option, *ansi_c* is used as the default target host language compiler.

cobpre – *mf_byte* — MicroFocus COBOL compiler with byte-aligned data. If you set the precompiler with this argument, you must set the MicroFocus COBOL compiler with the -C NOIBMCOMP argument.

mf_word — MicroFocus COBOL compiler with word-aligned data. If you set the precompiler with this argument, you must set the MicroFocus COBOL compiler with the -C IBMCOMP argument. If you do not use this option, *mf_byte* is used as the default compiler.

-D*database_name*

specifies the name of the database to parse against. Use this option when you want to do SQL semantic checking at precompile time. If -G is specified, a use *database* command will be added to the beginning of the *filename.sql* file. If you do not use this option, the precompiler uses your default database on the Adaptive Server.

-F*fips_level*

checks that the SQL contained in the Embedded SQL source file is ANSI SQL-89 or SQL-92E compliant. The user must provide an Adaptive Server name and a valid user name/password combination for the precompiler to connect to the server. The -F option generates code enabling the server to continue checking the SQL for compliance while an application is being run. Valid values are SQL89 or SQL92E.

-G[*isql_file_name*] (optional)

generates stored procedures for appropriate SQL statements and saves them to an isql file. You must execute isql on the isql file to load the stored procedure. If you have multiple input files, you may use -G, but you cannot specify an argument.

If you have multiple input files or do not specify the argument, the default target file name(s) will be the input file name(s) with the extension ".sql" appended (or replacing any input file name extension). Also, see option -T*tag_id* to specify tag identification for stored procedures. If you do not use the -G option, no stored procedures are generated.

-H[*server_name*] (optional)

invokes the HA failover option. The secondary server is used if the primary server fails. The command line argument parser reads the -H flag, stores the failover server name, and generates the necessary CT-Lib statements to implement HA failover. To use this option, the secondary server must be up and running and correctly listed in the interfaces file as an HA failover server.

-I*include_directory*

specifies a directory where Embedded SQL will search for *include* files. You may specify this option any number of times. Embedded SQL searches the directories in their command line order. If you do not use this option, the default include directories are *$SYBASE/include* and the current working directory.

-J*locale_for_charset*

specifies the character set of the source file that is being precompiled. The option's value must be a locale name that corresponds to 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.

To determine which character set to use as its default, the precompiler looks for a locale name. The precompiler searches for the following environment variables in the following order:

- LC_ALL
- LANG

If LC_ALL is defined, the precompiler uses its value as the locale name. If LC_ALL is not defined but LANG is defined, the precompiler uses its value as the locale name.

If neither LC_ALL nor LANG is defined, the precompiler uses a locale name of "default."

The precompiler looks up the locale name in the *locales* file, and uses the character set associated with it as the default character set.

-K*syntax_level*

specifies the level of syntax checking to perform. The choices are:

- NONE

  No checking (no Adaptive Server is required during precompiling).

- SYNTAX

  Syntax is checked by Adaptive Server during precompiling but does not require databases objects referred to by Embedded SQL statements to be present.

- SEMANTIC

  Embedded SQL statements checked for syntactic and semantic correctness—database objects referred to by statements must be present. Semantic checking is not performed on statements that include input host variables of type CS_TEXT or CS_IMAGE, regardless of command line options.

-L[*listing_file_name*] (argument is optional)
: generates one or more listing files. A listing file is a version of the input file with each line numbered and followed by any applicable error message. If you have multiple input files, you may use -L, but you cannot specify an argument.

If you have multiple input files or do not specify the argument, the default listing file name(s) will be the input file name(s) with the extension ".lis" appended (or replacing any input file name extension). If you do not use this option, no listing file is generated.

-N*interfaces_file_name*
: specifies an interfaces file name to the precompiler. If you do not use this option, the default interfaces file *$SYBASE/interfaces* is used.

-O*target_file_name*
: specifies the target or output file name. You cannot use this option if there are multiple input files (default target file names will be assigned). If you do not use this option, the default target file name will be the input file name with:

cpre – the extension *.c* appended (replacing any input file name extension).

cobpre – the extension *.cbl* appended (or replacing any input file name extension).

-P[*password*] (used with option -U*user_id*; argument is optional)
: specifies an Adaptive Server password for SQL syntax checking at precompile time. -P without an argument or with the key word NULL specifies a null ("") password. If you use option -U*user_id* but do not use -P, the precompiler prompts you to enter a password.

-S*server_name*
: specifies the name of the Adaptive Server for SQL syntax checking at precompile time. If you do not use this option, the default Adaptive Server name is taken from the DSQUERY environment variable.

-T*tag_id* (used with option -G)

specifies a tag identification (up to 3 characters) to append to the end of the generated stored procedure group name.

For example, if you type -Tdbg as part of your command, your generated stored procedures will be given the name of the input file with the tag identification *dbg* appended: *program_dbg;1*, *program_dbg;2,* and so on.

Programmers can use tag identification to test changes to an existing application without destroying the existing generated stored procedures, which may be in use. If you do not use this option, no tag identification is added to the stored procedure name.

-U*user_id*

specifies the Adaptive Server user identification.

This option allows you to check SQL syntax at precompile time. It causes the precompiler to pass SQL statements to the server for parsing only. If the server detects syntax errors, the errors are reported and no code is generated. If you use the -U option but not -P[*password*], Embedded SQL prompts you to enter a password.

If you do not use this option, the precompiler does not connect to a server or perform SQL syntax checking of the input file beyond what is required to generate the target file.

-V*version_number*

specifies the Client-Library version number:

cpre – the version number must be CS_VERSION_100 or higher and the value must exactly match one of the legal values of the version number argument to the ct_init function. (See the *Open Client Client-Library Reference Manual* for information about the ct_init function.

cobpre – for COBOL, the version number must match one of the values from *cobpub.cbl*.

If you do not use this option, the default is the most recent version of Client-Library available with the precompiler (CS_VERSION_110 for Open Client and Open Server version 11.1).

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

To determine which language and character set to use as its default for messages, the precompiler:

1. Looks for a locale name. The precompiler searches for the following environment variables in the following order:
   - LC_ALL
   - LANG:

     If LC_ALL is defined, the precompiler uses its value as the locale name. If LC_ALL is not defined but LANG is defined, the precompiler uses its value as the locale name.

     If neither LC_ALL nor LANG is defined, the precompiler uses a locale name of “default.”
2. Looks up the locale name in the *locales* file, and uses the language and character set associated with it as the default for messages.

program[.ext] [program[.ext]] . . .
is the input file name(s) of the Embedded SQL/C source program. You can enter as many input file names as you wish. The file name format and length can be anything you wish as long as it does not violate any operating system rules. See “Comments” following for information about target files and multiple input files.

@file_name
can be used to specify a file containing any of the above command line arguments. The precompiler reads the arguments contained in this file in addition to any arguments already specified. If the file specified with *@file_name* contains names of the files to precompile, place the argument at the end of the command line.

Examples

1. Run the precompiler (ANSI-compliant):

```
cobpre | cpre program.pco|.pc
```

2. Run the precompiler with generated stored procedures and FIPS flagging (ANSI-compliant):

```
cobpre | cpre -G -f program1.pco|.pc program2.pco|.pc
```

3. Run the precompiler for two input files with cursors open across transactions (not ANSI compliant):

```
cobpre | cpre -a program1.pco|.pc program2.pco|.pc
```

4 Display the precompiler version information only:

```
cobpre | cpre -v
```

Usage

DCE precomiler versions

Versions of the precompiler executables are provided for use on machines where DCE is installed.

You can run the regular executables on DCE machines, but they will not be able to connect to servers if DCE is configured as the default directory service for Sybase client applications. (The default directory service is determined by the setup of the *$SYBASE/install/libtcl.cfg* file. See the Open Client and Open Server *Configuration Guide* for UNIX for details).

- Optional arguments

  If you use -G[*isql_file_name*], -L[*listing_file_name*], or -P[*password*] without specifying an argument, another option or the keyword NULL or double quotes ("") must follow on the command line. You cannot follow these options with an input file name. If you do, the precompiler will mistake the input file name for the option argument.

- ANSI standards

  The cobpre | cpre command defaults are set up for ANSI standard behavior.

- The -a, -c, -d, -f, -m, and -r options affect only the connect statement. If your source file does not contain a connect statement or you use the -x option, these options have no effect.

- Target file

  cpre – The default target file name is *program.c*. If you have only one input file, you may use option -O*target_file_name* to specify a target file name. If you have multiple input files, the default target files will be named *first_input_file.c*, *second_input_file.c*, and so on.

  cobpre – The default target file name is the input file name with the extension ".cbl" (for Micro Focus COBOL) appended (or replacing any input file name extension). If you have only one input file, you may use option -O*target_file_name* to specify a target file name. If you have multiple input files, the default target files will be named *first_input_file.cbl*, *second_input_file.cbl*, and so on.

- Option format

Options will work with or without a space before the argument. For example, either of these options will work:

```
-Tdbg
```

or

```
-T dbg
```

- Multiple input files

  The precompiler can handle multiple input files. However, you cannot use the option -O*target_file_name*; you must accept the default target file names (see “Target file” above). If you use option -G[*isql_file_name*], you may not specify an argument; the default *isql* file names will be *first_input_file.sql, second_input_file.sql,* and so on. If you use option -L[*listing_file_name*], you may not specify an argument; the default listing file names will be *first_input_file.lis, second_input_file.lis,* and so on.

Developing an application

This section lists the steps most commonly used in developing an Embedded SQL application. You may need to adapt this process to meet your own requirements.

1. Run the precompiler with options -c, -D*database_name,* -P[*password*], -S*server_name*, and -U*user_id* for syntax checking and debugging. Do not use -G[*isql_file_name*]. Compile and link the program to make sure the syntax is correct.

2. Make all necessary corrections. Run the precompiler with options -D*database_name*, -P[*password*], -S*server_name*, -U*user_id*, -G[*isql_file_name*], and -T*tag_id* to generate stored procedures with tag ids for a test program. Compile and link the test program. Load the stored procedures with this command:

   ```
   isql -P[password] -Sserver_name -Uuser_id \
        <isql_file_name
   ```

   Run tests on your program.

3. Run the precompiler with options -D*database_name* , -P[*password*], -S*server_name*, -U*user_id*, and -G[*isql_file_name*] (but without option -T) on the corrected version of the program. Compile and link the program. Load the stored procedures with this command:

   ```
   isql -P[password] -Sserver_name -Uuser_id \
        <isql_file_name
   ```

   The final distribution program is ready to run.

cobpre | *cpre d*efaults

The following table lists the options and defaults for the cobpre and cpre utilities:

Table A-6: Defaults and options for cobpre and cpre defaults

Option	**Default if option not used**
-C*compiler*	cpre – The *ansi_c* compiler cobpre – The *mf_byte* compiler.
-D*database_name*	The default database on Adaptive Server.
-G[*isql_file_name*]	No stored procedures are generated.
-H[*server_name*]	Failover server name that is used if the primary server fails.
-I*include_directory*	Default directory is *$SYBASE/include*.
-J*locale_for_charset*	Platform-specific.
-L[*listing_file_name*]	No listing file is generated.
-O*target_file_name*	The default target filename is the input filename with the extension *.c* (for cpre) or *.cbl* (for cobpre) appended (or replacing any input filename extension).
-N*interfaces_file_name*	The *$SYBASE/interfaces* file.
-P[*password*]	You are prompted for a password unless you use -U*user_id*.
-S*server_name*	The default Adaptive Server name is taken from the DSQUERY environment variable.
-T*tag_id*	No tag IDs are added to the stored procedure names (generated with -G).
-U*user_id*	The precompiler does not connect to an Adaptive Server or do syntax-checking beyond what is required to generate the target file.
-V*version_number*	CS_VERSION_110 for version 11.1.
-Z*locale_for_messages*	Platform/environment specific.

defncopy

Description

Copies definitions for specified views, rules, defaults, triggers, or procedures from a database to an operating system file or from an operating system file to a database.

Note The defncopy utility cannot copy table definitions or reports created with Report Workbench™.

Syntax

```
defncopy
        [-v] [-X]
        [-a display_charset]
        [-I interfaces_file]
        [-J [client_charset]]
        [-K keytab_file]
        [-P password]
        [-R remote_server_principal]
        [-S [server]]
        [-U username]
        [-V [security_options]]
        [-z language]
        [-Z security_mechanism]
        {in filename dbname |    out filename dbname
        [owner.]objectname [[owner.]objectname...] }
```

Parameters

in | out
: specifies the direction of definition copy in relation to the database. For example, specifying "in" copies definitions into the database.

filename
: specifies the name of the operating system file destination or source for the definition copy. The copy out overwrites any existing file.

dbname
: specifies the name of the database to copy the definitions from or to.

objectname
: specifies name(s) of database object(s) for defncopy to copy out. Objects should not be specified when copying definitions into a database.

-a *display_charset*
: allows you to run defncopy from a terminal where the character set differs from that of the machine on which defncopy is running. -a in conjunction with -J specifies the character set translation file (*.xlt* file) required for the conversion. Use -a without -J only if the client character set is the same as the default character set.

-I *interfaces_file*
specifies the name and location of the interfaces file to search when connecting to Adaptive Server. If you do not specify -I, defncopy looks for an interfaces file located in the directory specified by the SYBASE environment variable.

-J *client_charset*
specifies the character set to use on the client. A filter converts input between *client_charset* and the Adaptive Server character set.

-J *client_charset* requests that Adaptive Server convert to and from *client_charset*, the client's character set.

-J with no argument sets character-set conversion to NULL. No conversion takes place. Use this if the client and server are using the same character set.

Omitting -J sets the character set to a default for the platform.The default may not necessarily be the character set that the client is using. (See the *System Administration Guide* and the *System Administration Guide Supplement* for more information about character sets and the associated flags.

Note The ascii_7 character set is compatible with all character sets. If either the Adaptive Server's or the client's character set is set to ascii_7, any 7-bit ASCII character is allowed to pass between client and server unaltered. Other characters produce conversion errors. Character-set conversion issues are covered more thoroughly in the *System Administration Guide*.

-K *keytab_file*
can be used only with DCE security. It specifies a DCE keytab file that contains the security key for the user name specified with -U option. Keytab files can be created with the DCE dcecp utility. See your DCE documentation for more information.

If the -K option is not supplied, the user of defncopy must be logged in to DCE with the same user name as specified with the -U option.

-P *password*
allows you to specify your password. This option is ignored if -V is used.

-R *remote_server_principal*
specifies the principal name for the server. By default, a server's principal name matches the server's network name (which is specified with the -S option or the DSQUERY environment variable). The -R option must be used when the server's principal name and network name are not the same.

-S*server*

specifies the name of the Adaptive Server to connect to. Without -S, defncopy looks for the server specified by your DSQUERY environment variable.

-U *username*

allows you to specify a login name. Login names are case sensitive. If you do not specify *username*, defncopy uses the current user's operating system login name.

-V *security_options*

specifies network-based user authentication. With this option, the user must log in to the network's security system before running the utility. In this case, users must supply their network user name with the -U option; any password supplied with the -P option is ignored.

-V can be followed by a *security_options* string of key-letter options to enable additional security services. These key letters are:

c – Enable data confidentiality service

i – Enable data integrity service

m – Enable mutual authentication for connection establishment

o – Enable data origin stamping service

r – Enable data replay detection

q – Enable out-of-sequence detection

-v

displays the version number and copyright message of defncopy and returns to the operating system.

-X

specifies that, in this connection to the server, the application initiate the login with client-side password encryption. defncopy (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which defncopy uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

If defncopy crashes, the system creates a core file which contains your password. If you did not use the encryption option, the password appears in plain text in the file. If you used the encryption option, your password is not readable.

-z *language*
specifies the official name of an alternate language that the server uses to display defncopy prompts and messages. Without the -z flag, defncopy uses the server's default language. Add languages to an Adaptive Server at installation, or afterwards with the utility langinstall or the stored procedure sp_addlanguage.

-Z *security_mechanism*
specifies the name of a security mechanism to use on the connection.

Security mechanism names are defined in the *$SYBASE/install/libtcl.cfg* configuration file. If no *security_mechanism* name is supplied, the default mechanism is used. For more information on security mechanism names, see the description of the *libtcl.cfg* file in the Open Client and Open Server *Configuration Guide* for UNIX.

Examples

```
defncopy -Usa -P -SMERCURY in new_proc stagedb
```

Copies definitions from the file *new_proc* into the database stagedb on server MERCURY. The connection with MERCURY is established with a user of name "sa" and a NULL password.

```
defncopy -S -z french out dc.out employees sp_calccomp
sp_vacation
```

Copies definitions for objects "sp_calccomp" and "sp_vacation" from the "employees" database on the SYBASE server to the file *dc.out*. Messages and prompts are displayed in "french." The user is prompted for a password.

Usage

- Invoke the defncopy program directly from the operating system. defncopy provides a non-interactive way of copying out definitions (create statements) for views, rules, defaults, triggers, or procedures from a database to an operating system file. Alternatively, it copies in all the definitions from a specified file.

  You must have select permission on the *sysobjects* and *syscomments* tables to copy out definitions; you do not need permission on the object itself.

- You must have the appropriate create permission for the type of object you are copying in. Objects copied in belong to the copier. A System Administrator copying in definitions on behalf of a user must log in as that user to give the user proper access to the reconstructed database objects.

- The in *filename* or out *filename* and the database name are required and must be unambiguously stated. For copying out, use filenames that reflect both the object's name and its owner.

- defncopy ends each definition that it copies out with the comment

```
/* ### DEFNCOPY: END OF DEFINITION  */
```

When assembling definitions in an operating system file to be copied into a database using defncopy, each definition must be terminated using the "END OF DEFINITION" string.

- Enclose values specified to defncopy in quotation marks if they contain characters that could be significant to the shell.

Warning! Long comments (more than 100 characters) placed before a create statement may cause defncopy to fail.

New features

defncopy for System 11 is built with Client-Library. It must be used if DCE is the default directory service for Sybase client applications or if you use DCE security services on bulk copy sessions. The defncopy user interface is unchanged except for the following:

- New command-line options have been added to enable network-based security services on the connection as follows:

  -K *keytab_file*
  -R *remote_server_principal*
  -V *security_options*
  -Z *security_mechanism*

- The -y *sybase_directory* option is ignored.

isql

Description

Interactive SQL parser to Adaptive Server.

Syntax

isql [-b] [-e] [-F] [-n] [-p] [-v] [-X] [-Y]
 [-a *display_charset*]
 [-A *size*]
 [-c *cmdend*]
 [-D *database*]
 [-h *headers*]
 [-H *hostname*]
 [-i *inputfilename*]
 [-I *interfaces_file*]
 [-J *client_charset*]
 [-K *keytab_file*]
 [-l *login_timeout*]

[-m *errorlevel*]
[-o *outputfilename*]
[-P *password*]
[-R *remote_server_principal*]
[-s *colseparator*]
[-S *server*]
[-t *timeout*]
[-U *username*]
[-V [*security_options*]]
[-w *columnwidth*]
[-z *language*]
[-Z *security_mechanism*]

Parameters

-a *display_charset*

allows you to run isql from a terminal where the character set differs from that of the machine on which isql is running. -a in conjunction with -J specifies the character set translation file (*.xlt* file) required for the conversion. Use -a without -J only if the client character set is the same as the default character set.

-A *size*

specifies the network packet size to use for this isql session. For example:

```
isql -A 2048
```

sets the packet size to 2048 bytes for this isql session. To check, enter:

```
select * from sysprocesses
```

The value is displayed under the *network_pktsz* heading.

size must be between the values of the default network packet size and maximum network packet size configuration variables, and must be a multiple of 512.

Use larger-than-default packet sizes to perform I/O-intensive operations, such as readtext or writetext operations.

Setting or changing Adaptive Server's packet size does not affect remote procedure calls' packet size.

-b – disables the display of the table headers output.

-c *cmdend*

resets the command terminator. By default, terminate commands and send them to Adaptive Server by typing "go" on a line by itself. When you reset the command terminator, do not use SQL reserved words or control characters. Make sure to escape shell meta-characters such as , ? ( ) [ ] $ and so on.

-D *database*
selects a database that the isql session begins in.

-e
echoes input.

-E *editor*
specifies an editor other than your default editor.

-F
enables the FIPS flagger. With this option, the Adaptive Server flags any nonstandard SQL commands sent.

-h *headers*
specifies the number of rows to print between column headings. The default prints headings only once for each set of query results.

-H *hostname*
sets the client host name.

-i *inputfilename*
specifies the name of an operating system file to use for input to isql. The file must contain command terminators ("go" by default).

Specifying the parameter as follows:

```
-i inputfile
```

is equivalent to:

```
< inputfile
```

If you use -i and do not specify your password on the command line, Adaptive Server prompts you for it. If you use < *inputfile* and do not specify your password on the command line, you must specify your password as the first line of the input file.

-I*interfaces_file*
specifies the name and location of the interfaces file to search when connecting to Adaptive Server. If you do not specify -I, isql looks for an interfaces file (*sql.ini* for Windows platforms) located in the *ini* directory that is below the directory specified by the SYBASE environment variable.

-J *client_charset*
specifies the character set to use on the client. -J *client_charset* requests that Adaptive Server convert to and from *client_charset*, the character set used on the client. A filter converts input between *client_charset* and the Adaptive Server character set.

-J with no argument sets character set conversion to NULL. No conversion takes place. Use this if the client and server use the same character set.

Omitting -J sets the character set to a default for the platform. The default may not necessarily be the character set that the client is using.

-K *keytab_file*
can be used only with DCE security. It specifies a DCE keytab file that contains the security key for the user name specified with -U option. Keytab files can be created with the DCE dcecp utility. See your DCE documentation for more information.

If the -K option is not supplied, the user of isql must be logged in to DCE with the same username as specified with the -U option.

-l *login_timeout*
specifies the maximum timeout value allowed when connecting to Adaptive Server. The default is 60 seconds. This value affects only the time that isql waits for the server to respond to a login attempt. To specify a timeout period for command processing, use the -t *timeout* parameter.

-m *errorlevel*
customizes the error message display. For errors of the severity level specified or higher only the message number, state, and error level display; no error text appears. For error levels lower than the specified level, nothing appears.

- *curread* (current read level) is the initial level of data that you can read during this session. *curread* must dominate *curwrite.*
- *curwrite* (current write level) is the initial sensitivity level that is applied to any data that you write during this session.
- *maxread* (maximum read level) is the maximum level at which you can read data. This is the upper bound to which you as a multilevel user can set your *curread* during the session. *maxread* must dominate maxwrite.
- *maxwrite* (maximum write level) is the maximum level at which you can write data. This is the upper bound to which you as a multiuser can set your *curwrite* during a session. *maxwrite* must dominate *minwrite* and *curwrite*.

- *minwrite* (minimum write level) is the minimum level at which you can write data. This is the lower bound to which you as a multiuser can set *curwrite* during a session. *minwrite* must be dominated by *maxwrite* and *curwrite*.
- *label_value* is the actual value of the label, expressed in the human-readable format used on your system (for example, “Company Confidential Personnel”).

-n
removes numbering and the prompt symbol (>) from input lines.

-o *output_filename*
specifies the name of an operating system file to store the output from isql.

```
-o outputfile
```

which is similar to:

```
> outputfile
```

-p
prints performance statistics.

-P *password*
specifies your current Adaptive Server password. This option is ignored if -V is used. Passwords are case sensitive and can be from 6 to 30 characters in length.

-R *remote_server_principal*
specifies the principal name for the server. By default, a server’s principal name matches the server’s network name (which is specified with the -S option or the DSQUERY environment variable). The -R option must be used when the server’s principal name and network name are not the same.

-s *colseparator*
resets the column separator character, which is blank by default. To use characters that have special meaning to the operating system (for example, |, ;, &, <, >), enclose them in quotes or precede them with a backslash.

-S *server*
specifies the name of the Adaptive Server to connect to. Without -S, isql looks for the server specified by your DSQUERY environment variable.

-t *timeout*
specifies the number of seconds before a command times out. If you do not specify a timeout, a command runs indefinitely. This affects commands issued from within isql, not the connection time.

-U *username*
specifies a login name. Logins are case sensitive.

-V *security_options*
specifies network-based user authentication. With this option, the user must log in to the network's security system before running the utility. In this case, users must supply their network user name with the -U option; any password supplied with the -P option is ignored.

-V can be followed by a *security_options* string of key-letter options to enable additional security services. These key letters are:

c – Enable data confidentiality service

i – Enable data integrity service

m – Enable mutual authentication for connection establishment

o – Enable data origin stamping service

q – Enable out-of-sequence detection

r – Enable data replay detection

-v
prints the version and copyright message of the isql software that you are using.

-w *columnwidth*
sets the screen width for output. The default is 80 characters. When an output line reaches its maximum screen width, it breaks into multiple lines.

-X
initiates the login connection to the server with client-side password encryption. isql (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which isql uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

If isql crashes, the system creates a core file which contains your password. If you did not use the encryption option, the password appears in plain text in the file. If you used the encryption option, your password is not readable.

-Y
tells the Adaptive Server to use chain transactions.

-z *language*
is the official name of an alternate language to display isql prompts and messages. Without -z, isql uses the server's default language. Add languages to an Adaptive Server at installation, or afterwards with the utility langinstall or the stored procedure sp_addlanguage.

-Z *security_mechanism*
specifies the name of a security mechanism to use on the connection.

Security mechanism names are defined in the *libtcl.cfg* configuration file located in the *ini* subdirectory below the Sybase installation directory. If no *security_mechanism* name is supplied, the default mechanism is used. For more information on security mechanism names, see the description of the *libtcl.cfg* file in the Open Client and Open Server *Configuration Guide* for UNIX Platforms.

Examples

Example 1

```
isql
Password:

 1>select *
 2>from authors
 3>where city = "Oakland"
4>go
```

Executes the command.

Example 2

```
isql -Ujoe -Pabracadabra
1>select *
2>from authors
3>where city = "Oakland"
4>vi
```

Lets you edit the query. When you write and save the file, you are returned to isql. The query is displayed. Type go to execute it.

Example 3

```
isql -U alma Password:
1>select *
2>from authors
3>where city = "Oakland"
4>reset
5>quit
```

reset clears the query buffer. quit returns you to the operating system.

Usage

- To use isql interactively, give the command isql (and any of the optional flags) at your operating system prompt. The isql program accepts SQL commands and sends them to Adaptive Server. The results are formatted and printed on standard output. Exit isql with quit or exit.
- Terminate a command by typing a line beginning with the default command terminator go or other command terminator if the -c option is used. You may follow the command terminator with an integer to specify how many times to run the command. For example, to execute this command 100 times, type:

  ```
  select x = 1
   go 100
  ```

  The results display once at the end of execution.
- If you enter an option more than once on the command line, isql uses the last value. For example, if you enter the following command:

  ```
  isql -c. -csend
  ```

  "send," the second value for -c, overrides ".", the first value. This allows you to override any aliases you set up.
- To call an editor on the current query buffer, enter its name as the first word on a line. Define your preferred callable editor by specifying in with the EDITOR environment variable. If EDITOR is undefined, the default is vi.

  Execute operating system commands by starting a line with "!!" followed by the command. Call alternate editors this way, without defining EDITOR.
- To clear the existing query buffer, type reset on a line by itself. isql discards any pending input. You can also press Ctrl-c anywhere on a line to cancel the current query and return to the isql prompt.
- Read in an operating system file containing a query for execution by isql as follows:

  ```
  isql -U alma -P****** < input_file
  ```

  The file must include command terminator(s). The results appear on your terminal. Read in an operating system file containing a query and direct the results to another file as follows:

  ```
  isql -U alma -P****** < input_file > output_file
  ```

- Case is significant for the isql flags.

- isql displays only six digits of float or real data after the decimal point, rounding off the remainder.
- When using isql interactively, read an operating system file into the command buffer with the following command:

  ```
  :r filename
  ```

  Do not include a command terminator in the file; enter the terminator interactively once you have finished editing.
- You can include comments in a Transact-SQL statement submitted to Adaptive Server by isql. Open a comment with "/*". Close it with "*/" as the following example demonstrates:

  ```
  select au_lname, au_fname
   /*retrieve authors' last and first names*/
   from authors, titles, titleauthor
   where authors.au_id = titleauthor.au_id
   and titles.title_id = titleauthor.title_id
   /*this is a three-way join that links authors
  **to the books they have written.*/
  ```

  If you want to comment out a go command, it should not be at the beginning of a line. For example:

  ```
  /*
  **go
  */
  ```

  should be used to comment out the go command instead of:

  ```
  /*
  go
  */
  ```

isql

isql is built with Client-Library for better performance. isql_r is the same as isql and must be used if DCE is the default directory service for Sybase client applications or if you use DCE security services on bulk copy sessions.

The isql user interface is unchanged except for:

- The 5701 ("changed database") server message is no longer displayed after login or after issuing a use database command.
- There are two new optional flags:

  -b – disables column headers from printing
  -D *database* – selects the start-up database that isql uses

- The following command-line options have been added to enable network-based security services on the connection:

  -K *keytab_file*
  -R *remote_server_principal*
  -V *security_options*
  -Z *security_mechanism*

- Error message format is different than previous versions of isql. If you have scripts that perform routines based on the values of these messages, you may need to rewrite them.
- The -y *sybase_directory* option has been removed.

Additional commands within *isql*:

Table A-7: isql session commands

Command	Description
reset	Clears the query buffer
quit or exit	Exits from isql
vi	Calls the editor
!! *command*	Executes an operating system command

See also

sp_addlanguage, sp_addlogin, sp_configure, sp_defaultlanguage, sp_droplanguage, and sp_helplanguage in the Adaptive Server Enterprise *Reference Manual*.

APPENDIX B

Environment Variables

This appendix contains the values of the environment variables required for your Sybase applications to compile and work correctly. The environment variables that must be set depends on your application, and include:

- SYBASE – set to the path of the Sybase installation directory.
- SYBASE_OCS – set to the subdirectory containing the Open Client and Open Server version number. For example, *OCS-12_5*.
- DSQUERY – set to the name of the Adaptive Server or Open Server.
- DSLISTEN – set to the name of the Open Server.
- SYBPLATFORM – depends on the platform that you are running and whether or not you are using reentrant libraries. Refer to Table B-1 for the appropriate variable setting.
- You must set the platform specific library path variable listed in Table B-1 to *$SYBASE/$SYBASE_OCS/lib* to run programs linked with shareable (dynamic) libraries. If you are running in debug mode, set the platform-specific library path variable to *$SYBASE/$sybase_ocs/devlib*.

  For ESQL/COBOL applications, include the location of the *$COBDIR/coblib* directory.

Table B-1: SYBPLATFORM and library path

Platform	SYBPLATFORM setting	Platform-specific library path variable
HP Tru64 UNIX	axposf	LD_LIBRARY_PATH
using native threads	nthread_axposf	
HP Itanium - 32-bit	hpia	LD_LIBRARY_PATH
HP Itanium - 32-bit using native threads	nthread_hpia	
HP Itanium - 64-bit	hpia64	
HP Itanium - 64-bit using native threads	nthread_hpia64	
HP 9000(8xx) - 32-bit	hpux	SHLIB_PATH
HP 9000(8xx) - 32-bit using native threads	nthread_hpux	LD_LIBRARY_PATH
HP 9000(8xx) - 64-bit	hpux64	
HP 9000(8xx) - 64-bit using native threads	nthread_hpux64	
IBM RS/6000 - 32-bit	rs6000	LIBPATH
IBM RS/6000 - 32-bit using native threads	nthread_rs6000	
IBM RS/6000 - 64-bit	rs600064	
IBM RS/6000 - 64-bit using native threads	nthread_rs600064	
IBM AIX 5.x 64-bit	ibmaix64	
Linux - 32-bit using native threads	nthread_linux	LD_LIBRARY_PATH
Linux - 64 - bit	linux64	
Linux - 64-bit using native threads	nthread_linux64	
SGI – 32-bit version	sgi	LD_LIBRARY_PATH
SGI – 64-bit version	sgi64	
SGI – 32-bit version using native threads	nthread_sgi	
SGI – 64-bit version using native threads	nthread_sgi64	

Platform	SYBPLATFORM setting	Platform-specific library path variable
Sun Solaris 2.x - 32-bit	sun_svr4	LD_LIBRARY_PATH
Sun Solaris 2.x - 32-bit using native threads	nthread_sun_svr4	
Sun Solaris 2.x - 64-bit	sun_svr464	
Sun Solaris 2.x - 64-bit using native threads	nthread_sun_svr464	

For Embedded SQL/COBOL applications you must set the following environment variables in addition to the ones listed above:

- COBDIR – set to the path of your COBOL compiler
- PATH – add *$COBDIR/bin*

Index