SYBASE®

Programmer’s Supplement

Open Client™ and Open Server™

12.5.1

MICROSOFT WINDOWS

DOCUMENT ID: DC35455-01-1251-01

LAST REVISED: November 2003

Contents

About This Book

This book supplements the Open Client™ and Open Server™ reference manuals and programmer's guide. It provides the platform-specific information you need to create, configure for, and troubleshoot applications using Open Client and Open Server products for the following Microsoft Windows platforms:

- Windows NT
- Windows 2000
- Windows 2003
- Windows XP

From this point on, in this document, references to all Microsoft Windows platforms will be referred to as "Windows," except where noted otherwise.

Audience

The primary audiences for this book are:

- Desktop application developers who create Sybase® or third-party applications using Open Client and Open Server products
- Anyone who needs information about the bcp, defncopy, isql, wbcp, wdefncopy, wdllvers, and wisql utilities
- Anyone who needs information about the cpre and cobpre precompilers.

Related documents

Each Open Client and Open Server product has its own set of user documentation. The following table 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*

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

See the Open Client and Open Server *Configuration Guide* for Microsoft Windows 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 are:

Table 2: Syntax conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are lowercase bold.
variable	Variables, or words that stand for values that you fill in, are in italics.

Key	Definition
`{ }`	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, separating your choices with commas to be typed as part of the command.

The following examples illustrate the syntax conventions described above.

Vertical bars indicates that you choose one and only one option within the *curly braces*.

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

Commas indicates that you 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 don't have to choose any of the items in square brackets.

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

```
[anchovies]
```

Vertical bars within square brackets indicates that you can choose none or only one within the *square brackets*.

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

Commas within *square brackets* indicates that 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 an *Ellipsis (...)* indicates that you can repeat the last unit as many times as you like. In the following syntax paradigm, 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, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Building Open Client and Open Server Applications

This chapter provides the information you need to start building applications using Open Client and Open Server libraries on the Windows platforms. It also describes the requirements for building Windows executables using Sybase libraries.

Open Client and Open Server requirements

Before you compile and link Open Client and Open Server applications on Windows platforms, you must:

- Have an ANSI-compliant C compiler installed
- Define the INCLUDE environment variable
- Define the LIB environment variable
- Set the PATH variable to include the *dll* subdirectory
- Have at least one Open Client Net-Library driver for Windows to make a network connection to the server

Note Sybase libraries for Windows platforms are designed for Win32 applications.

C compilers

You must have an ANSI-compliant C compiler installed if you plan to use the example programs or to build applications. Sybase has certified Microsoft's Visual C++ Version 5.0. Sybase may certify other compilers. For a list of currently certified compilers, check with your Sybase sales representative.

Compile and run an Open Client and Open Server program in the same way as any other C language program. (Refer to the instructions provided with your compiler for compiling and linking your application.)

Warning! If you have problems using an ANSI-compliant C compiler that Sybase has not certified, you can receive Sybase technical support only if the problems can be reproduced using a Sybase-certified compiler.

Client-Library compatibility

Client-Library version 12.5 on Windows is certified to work with the Open Server™, and Sybase Adaptive Server® products shown in Table 1-1:

Table 1-1: Open Client compatibility

Open Client 12.5 platform	Open Server 12.5.1	Open Server 12.5	Open Server 12.0	Adaptive Server 12.5.1	Adaptive Server 12.5	Adaptive Server 12.0
Windows NT 4.0 Service Pack 6 or later	x	x	x	x	x	x
Windows 2000, 2003, and XP	x	x	n/a	x	x	n/a

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

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

- For Windows NT, Sybase SQL Toolset™ versions 4.x and 5.0 (APT-Library™ and Report-Library applications) are not compatible with Open Client/C version 12.0 libraries.

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

Open Server compatibility

Open Server version 12.5 on Windows platforms is certified to work with the Client-Library/C and Adaptive Server products shown in Table 1-2:

Table 1-2: Open Server compatibility

Open Server 12.5 platform	**Client-Library 12.5.1**	**Client-Library 12.5**	**Client-Library 12.0**	**Adaptive Server 12.5.1**	**Adaptive Server 12.5**	**Adaptive Server 12.0**
Windows NT 4.0 Service Pack 6 or later	x	x	x	x	x	x
Windows 2000, 2003, and XP	x	x		x	x	

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

In addition, note these compatibility issues for Open Server:

- Header files included in an application must be the same version level as the library with which the application is linked.
- Bulk-Library routines cannot be used in applications that call Open Server version 2.x routines.
- DB-Library/C 11.x and later is no longer supported with Open Server 11.x and later.

Environment variables and header files

For your applications to function properly, you must set a number of environment variables. Table 1-3 lists descriptions of the required environment variables:

Table 1-3: Description of environment variables

Variable	Description
INCLUDE	Must contain the directory path that points to the *include* subdirectory in the Sybase installation directory. This subdirectory stores the header files when installation is complete.
LIB	Must contain the directory path that points to the *lib* subdirectory in the Sybase installation directory. This subdirectory stores the import library files once installation is complete.
PATH	Must contain the directory path that points to the *dll* subdirectory of the Sybase installation directory. This subdirectory stores the Sybase DLLs once installation is complete.

Header files

Table 1-4 lists header files that you need to include when compiling your Open Client and Open Server applications.

Table 1-4: Open Client and Open Server header files

DB-Library header files	Client-Library header files	Server-Library header files
sybdb.h – contains definitions and type definitions for use with DB-Library routines. Use *sybdb.h* only as documented in the Open Client *DB-Library/C Reference Manual*. The *sybdb.h* file includes all other required header files.	*ctpublic.h* – required by all Client-Library applications. This file includes all other required header files.	*ospublic.h* – required by all Server-Library applications. This file includes all other required header files.
ssybfront.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. *sybfront.h* also includes type definitions for datatypes that can be used in program variable declaration. *ssyberror.h* – contains error severity values and should be included if the program refers to those values.	*bkpublic.h* – required if your application makes calls to Bulk-Library. This file includes all other required header files.	*bkpublic.h* – required if your application makes calls to Bulk-Library. This file includes all other required header files.

Import libraries and Dynamic Link Libraries (DLLs)

This section describes import libraries and Dynamic Link Libraries.

Import libraries

The Open Client and Open Server import libraries contain information used by the linker to build Open Client and Open Server applications. Table 1-5 lists the import libraries you should include when compiling and linking your application:

Table 1-5: Open Client and Open Server import libraries

DB-Library import libraries	Client-Library import libraries	Server-Library import libraries
libsybdb.lib – DB-Library	*libct.lib* – Client-Library	*libsrv.lib* – Server-Library
libcomn.lib – An internal shared utility library	*libcs.lib* – CS-Library	*libct.lib* – Client-Library
libtcl.lib – Net-Library™	*libblk.lib* – Bulk-Library	*libcs.lib* – CS-Library
libintl.lib – Localization support library	You need to link with the Bulk-Library import library *libblk.lib* only if you make Bulk-Library calls.	*libblk.lib* – Bulk-Library
		You need to link with the Bulk-Library import library *libblk.lib* only if you make Bulk-Library calls.

Dynamic link libraries (DLLs)

At runtime, Windows Open Client and Open Server library applications must be able to call functions in the Open Client DLLs. Make sure that the Sybase DLLs are in your path. The PATH environment variable must contain the *dll* subdirectory of the Sybase installation directory. Table 1-6 lists the DLLs that are supplied with Open Client and Open Server libraries:

Table 1-6: Open Client and Open Server DLLs

DB-Library DLLs	Client-Library DLLs	Server-Library DLLs
libsybdb.dll - DB-Library	*libct.dll* - Client-Library	*libct.dll* - Client-Library
libintl.dll - Localization support library	*libcs.dll* - CS-Library	*libcs.dll* - CS-Library
libtcl.dll - Transport control layer	*libintl.dll* - Localization support library	*libintl.dll* - Localization support library
libcomn.dll - Internal common library	*libtcl.dll* - Transport control layer	*libtcl.dll* - Transport control layer
	libcomn.dll - Internal common library	*libcomn.dll* - Internal common library
	libblk.dll – Bulk-Library	*libsrv.dll* – Server-Library
		libblk.dll – Bulk-Library

Configuration requirements

For the example programs and your applications to run properly, you must meet these configuration and system requirements:

- The SYBASE environment variable must be defined.
- The *sql.ini* file must have a query entry for the server name used by Open Client applications.
- The *sql.ini* file must have a query entry for the server name used by Open Server applications.
- You should have a minimum of 64MB of memory for Windows platforms.

Note For information on setting the SYBASE environment variable and configuring the *sql.ini* file, see the Open Client and Open Server *Configuration Guide* for Microsoft Windows.

Platform-specific default values

Table 1-7 lists Open Client and Open Server properties with platform-specific default values:

Table 1-7: Client-Library platform-specific properties

Library	Property name	Description	Default value on Windows
Client-Library and Server-Library	CS_IFILE	The path and name of the *sql.ini* file.	The *sql.ini* file in the *ini* subdirectory defined by the SYBASE environment variable.
	CS_MAX_CONNECT	The maximum number of connections for this context.	25
	CS_PACKETSIZE	The TDS packet size.	512 bytes

Library	Property name	Description	Default value on Windows
DB-Library	DBSETFILE	The path and name of the *sql.ini* file.	The *sql.ini* file in the *ini* subdirectory defined by the SYBASE environment variable.
	DBSETMAXPROS	The maximum number of connections for this context.	25

Client-Library programming issues

This section explains the differences between the way certain Client-Library routines behave on Windows platforms and how they are documented in the Open Client *DB-Library/C Reference Manual* and the Open Client *DB-Library/C Programmer's Guide*.

ct_callback

Any Client-Library application routine that is registered with Client-Library using ct_callback must be declared as CS_PUBLIC and must be exported in the *.def* file. See the routine ex_clientmsg_cb in *exutils.c* in the sample directory for an example.

Using the debug DLLs

Depending upon options selected during installation, you can install both Client-Library debug and non-debug versions of *libct.dll*. The debug version of the DLL is stored in the *debug* subdirectory of the Sybase *dll* directory and the non-debug version in the *nondebug* subdirectory. Copy the version you want to use to the *dll* subdirectory of the Sybase installation directory. The application automatically uses the DLLs in the *dll* subdirectory of the Sybase installation directory. This ct_debug utility works only when you use the debug version of *libct.dll*.

Refer to the Open Client *DB-Library/C Reference Manual* for general information about the debug version of Client-Library.

Online help

Open Client for Windows platforms provides online help for the C versions of Client-Library, CS-Library, and Bulk-Library. This facility consists of:

- C routine reference information, including routine syntax, routine returns, example code fragments, and comments associated with each routine
- Programming topics associated with each of the three libraries

If you installed the online help facility when you ran setup, the help executable is located at *\help\ctlib.hlp* beneath the Sybase installation directory. You can access Client-Library help in one of these ways:

- Double-click on the file *ctlib.hlp* displayed in the File Manager.
- From the File Manager File menu, choose Run. Type the name of the help file at the Command Line prompt, and click OK.

Multithreaded support

Client-Library version 11.1 and later supports Windows platforms thread libraries for developing multithreaded applications. For an overview of developing multithreaded applications, refer to the Open Client *DB-Library/C Reference Manual*.

Example compile-and-link operations

This section shows an example *makefile* for a Windows Client-Library application for use by a Microsoft Visual C/C++ compiler, version 4.0:

```
####################################################################
# Microsoft makefile for sample programs
#
###################################################################
MAKEFILE=MAKEFILE

!ifndef SYBASE
SYBASEHOME=c:\sybase
!else
SYBASEHOME=$(SYBASE)
!endif

COMPILE_DEBUG = 1
```

```
# Compiler AND linker flags
!ifdef COMPILE_DEBUG
CFLAGS = /W3 /MD /nologo /Zi /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
!else
CFLAGS = /W3 /MD /nologo /Od /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE
!endif

ASYNCDEFS = -DUSE_SIG_HANDLER=0

HDRS = example.h exutils.h
MTHDRS = example.h thrdutil.h thrdfunc.h

#
# Where to get includes and libraries
#
# SYBASE is the environment variable for sybase home directory
#
SYBINCPATH =     $(SYBASEHOME)\$(SYBASE_OCS)\include
BLKLIB =      $(SYBASEHOME)\$(SYBASE_OCS)\lib\libblk.lib
CTLIB =              $(SYBASEHOME)\$(SYBASE_OCS)\lib\libct.lib
CSLIB =              $(SYBASEHOME)\$(SYBASE_OCS)\lib\libcs.lib
SYSLIBS =     kernel32.lib advapi32.lib msvcrt.lib

# The generalized how to make an .obj rule
.c.obj:
    cl /I. /I$(SYBINCPATH) $(ASYNCDEFS) $(CFLAGS) -Fo$@ -c $<

all: exasync compute csr_disp getsend rpc blktxt i18n multthrd usedir firstapp
exconfig secct wide_rpc wide_dynamic wide_curupd wide_compute

uni: uni_firstapp uni_csr_disp uni_compute uni_blktxt uni_rpc

exasync compute csr_disp getsend rpc blktxt i18n multthrd usedir firstapp
exconfig secct twophase: $*.exe
           @echo Sample '$*' was built

wide_rpc wide_dynamic wide_curupd wide_compute: $*.exe
           @echo Sample '$*' was built

uni_firstapp uni_csr_disp uni_compute uni_blktxt uni_rpc: $*.exe
           @echo Sample '$*' was built

sample.exe: sample.obj $(MAKEFILE)
```

```
    link $(LFLAGS) -out:$*.exe sample.obj $(SYSLIBS)

exasync.exe: ex_alib.obj ex_amain.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe ex_alib.obj ex_amain.obj exutils.obj $(SYSLIBS)
$(CTLIB)             $(CSLIB)

compute.exe: compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

wide_curupd.exe: wide_curupd.obj exutils.obj wide_util.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_util.obj $(SYSLIBS)
$(CTLIB) $(CSLIB)

wide_dynamic.exe: wide_dynamic.obj exutils.obj wide_util.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_util.obj $(SYSLIBS)
$(CTLIB) $(CSLIB)

wide_compute.exe: wide_compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_compute.obj $(SYSLIBS)
$(CTLIB) $(CSLIB)

exconfig.exe: exconfig.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

firstapp.exe: firstapp.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

csr_disp.exe: csr_disp.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

getsend.exe: getsend.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

rpc.exe: rpc.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

wide_rpc.exe: wide_rpc.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

blktxt.exe: blktxt.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)
$(BLKLIB)

i18n.exe: i18n.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)
```

```
multthrd.exe: multthrd.obj thrdfunc.obj thrdutil.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj thrdfunc.obj thrdutil.obj $(SYSLIBS)
$(CTLIB) $(CSLIB)

usedir.exe: usedir.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

secct.exe: secct.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_firstapp.exe: uni_firstapp.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_csr_disp.exe: uni_csr_disp.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_compute.exe: uni_compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_blktxt.exe: uni_blktxt.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)
$(BLKLIB)

uni_rpc.exe: uni_rpc.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

twophase.exe: twophase.obj ctpr.obj ctxact.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj ctpr.obj ctxact.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)
clean:
           -del *.obj
           -del *.map
           -del *.exe
           -del *.err
           -del *.ilk
           -del *.pdb
```

There are a few things to note in this example:

- INTEL libraries are represented.
- Sybase libraries are made for Win32 applications.
- SUBSYSTEM:CONSOLE indicates a console application.

Refer to the appropriate compile-and-link Microsoft documentation for additional information.

DB-Library programming issues

This section explains the differences between how certain DB-Library routines behave on Windows platforms and how they are documented in the Open Client *DB-Library/C Reference Manual*.

Compile-and-link line examples

The general form of the command to compile and link a DB-Library/C application is:

```
!ifdef COMPILE_DEBUG
 CFLAGS = /W3 /MD /nologo /Z7 /DWIN32
 LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
 !else
 CFLAGS = /W3 /MD /nologo /Od
 LFLAGS= /MAP /SUBSYSTEM:CONSOLE
 !endif
```

Server-Library programming issues

This section explains the differences between how certain Server-Library routines behave on Windows and how they are documented in the Open Server *Server-Library/C Reference Manual*.

srv_callback

Any Server-Library application routine that is registered with Server-Library using srv_callback must be declared as CS_PUBLIC. See the routine cs_err_handler in *utils.c* in the sample directory for an example.

Scheduling modes

Server-Library applications running under Windows can operate in either co-routine or preemptive scheduling mode. Co-routine (the default) scheduling is compatible with other platforms that do not support preemptive scheduling. To choose preemptive scheduling mode set the SRV_PREEMPT option to "true" using the srv_config function.

Preemptive mode programming overview

Under the preemptive scheduling mode, all threads are executable at the same time and are handled by the Windows scheduler. Preemptive scheduling prevents a single thread from monopolizing the server. When running your application under preemptive mode, your application can use the debugger's thread facility to manipulate threads. It can also perform blocking operations without bringing the server to a halt. Preemptive mode offers better performance for applications running under Windows because synchronizing threads incurs no overhead.

Note To guarantee portability to platforms where only co-routine scheduling is available, always protect global data using Server-Library's mutex facility rather than using the Windows-specific semaphore APIs.

This section offers information about Windows-specific preemptive programming using the srv_sleep and srv_wakeup calls.

srv_sleep

This code fragment illustrates the use of srv_sleep in preemptive mode:

```
/*
 ** Request the mutex to prevent the logging service
 ** from calling srv_wakeup before srv_sleep is called.
 */
 if (WaitForSingleObject(Mutex,INFINITE) != WAIT_OBJECT_0)
         return(CS_FAIL);
 /*
 ** Send the log_request to the logging service.
 */
 if (srv_putmsgq(log_service,log_request, SRV_M_NOWAIT) == CS_FAIL)
         return(CS_FAIL);
```

```
/*
** Sleep until the log service has processed the log request.
*/
srv_sleep(log_request, LOGWAIT, NULL, NULL, (CS_VOID*)Mutex, (CS_VOID*)0);

/*
** Return the log sequence number to the caller.
*/
return(CS_SUCCEED);
```

srv_wakeup

When srv_sleep is used in preemptive mode using a mutex, the corresponding srv_wakeup routine must be preceded by a request for the same mutex. This process ensures that the sleeping thread is ready for srv_wakeup to be executed. The following code fragment shows how srv_wakeup must be preceded by a request for the mutex when it is used in preemptive mode:

```
/*
 ** Loop forever, logging language text. srv_getmsg will cause
 ** this thread to be suspended until a message is available on
 ** the log_request message queue.
 */
 while((get_status = srv_getmsgq(msgqid, &log_request,
         SRV_M_WAIT, &info)) == CS_SUCCEED)
 {
      /*
      ** Do the logging here.
     */

     /*
     ** Request the mutex to make sure the sender
     ** has called srv_sleep.
     */
     if (WaitForSingleObject(Mutex,INFINITE) != WAIT_OBJECT_0)
             return(CS_FAIL);

         /*
         ** Wake up the thread that is waiting for the language
         ** text to be logged.
         */
         srv_wakeup(log_request, SRV_M_WAKE_FIRST, (CS_VOID*)0, (CS_VOID*)0);

     /*
     ** Release the mutex.
     */
     if (!ReleaseMutex(Mutex))
             return(CS_FAIL);
 }
```

Example of compile-and-link operations

The following example shows a *makefile* for compiling and linking a Windows 32-bit application:

```
############################################################################

#

# Microsoft makefile for building Sybase Open Server Samples for Windows NT

#

#############################################################################

MAKEFILE=MAKEFILE

!ifndef SYBASE
YBASEHOME=c:\sybase
!else
YBASEHOME=$(SYBASE)\$(SYBASE_OCS)
!endif

COMPILE_DEBUG = 1
# Compiler AND linker flags
!ifdef COMPILE_DEBUG
CFLAGS = /W3 /MD /nologo /Z7 /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
!else
CFLAGS = /W3 /MD /nologo /Od /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE
!endif
SYSLIBS = kernel32.lib advapi32.lib msvcrt.lib
SYBASELIBS = $(SYBASEHOME)\lib\libcs.lib $(SYBASEHOME)\lib\libct.lib
$(SYBASEHOME)\lib\libsrv.lib
BLKLIB = $(SYBASEHOME)\lib\libblk.lib
DBLIB = $(SYBASEHOME)\lib\libsybdb.lib
CTOSOBJ =     args.obj  attn.obj  bulk.obj \
                 connect.obj  ctos.obj  cursor.obj \
                 dynamic.obj  error.obj events.obj \
                 language.obj mempool.obj  options.obj \
                 params.obj \
                  rgproc.obj results.obj  rpc.obj \
                  send.obj  shutdown.obj
all: lang fullpass ctos regproc ctwait version intlchar osintro multthrd secsrv
lang fullpass ctos regproc ctwait version intlchar osintro multthrd secsrv:
$*.exe
           @echo Sample '$*' was built
# The generalized how to make an .obj rule
.c.obj:
    cl /I. /I$(SYBASEHOME)\include $(CFLAGS) -Fo$@ -c $<
lang.exe: lang.obj utils.obj
    link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

fullpass.exe: fullpass.obj utils.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

ctos.exe: $(CTOSOBJ)
       link $(LFLAGS) -out:$*.exe $(CTOSOBJ) $(SYSLIBS) $(SYBASELIBS) $(BLKLIB)

regproc.exe: regproc.obj utils.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

ctwait.exe: ctwait.obj ctutils.obj
       link $(LFLAGS) -out:$*.exe $*.obj ctutils.obj $(SYSLIBS) $(SYBASELIBS)

version.exe: version.obj ctutils.obj
       link $(LFLAGS) -out:$*.exe $*.obj ctutils.obj $(SYSLIBS) $(SYBASELIBS)

intlchar.exe: intlchar.obj utils.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj
$(SYSLIBS) $(SYBASELIBS)

osintro.exe: osintro.obj utils.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

multthrd.exe: multthrd.obj utils.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

secsrv.exe: secsrv.obj utils.obj secargs.obj
       link $(LFLAGS) -out:$*.exe $*.obj utils.obj secargs.obj $(SYSLIBS)
$(SYBASELIBS)

clean:
           -del *.obj
           -del *.map
           -del *.exe
           -del *.err
```

```
/*
```

Note that in this example:

- INTEL libraries are represented.
- Sybase libraries are made for Win32 applications.
- SUBSYSTEM:CONSOLE indicates that this is a console application.

Refer to the appropriate compile-and-link Microsoft documentation for additional information.

CHAPTER 2

Client Library/C Example Programs

Open Client Client-Library is a collection of routines used to write client applications. The example programs in this chapter demonstrate Client-Library and CS-Library functions. This chapter covers the following topics:

Client-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.

CS-Library, which is included with Open Client, is a collection of utility routines that you can use to write either an Open Client or an Open Server application. 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.

How to use the example programs

The example programs demonstrate specific Client-Library/C functionality. These programs are designed as guides for application programmers, not as Client-Library/C training aids. Read the descriptions at the top of each source file and examine the source code before attempting to use the example programs.

These simplified programs are not intended for use in a production environment. Programs written for a production environment need additional error-handling and special case-handling code.

Before you begin

Before you use the Open Client examples:

- Set the SYBASE environment variable to the path of the Sybase release directory (if it has not already been set).
- Set the DSQUERY environment variable to the server name (*server_name*) to which you want to connect.
- Use *makefile* to produce a sample executable named *example_name*.

For detailed information regarding configuration of your environment and variables, refer to the Open Client and Open Server *Configuration Guide for Microsoft Windows*.

Location of the example programs

The Client-Library example programs are located in the *sample\ctlib* subdirectory of the Sybase release directory. This directory contains:

- Online source code for the example programs
- Data files for the examples
- The header file, *example.h*, for the examples

Note Create a backup copy of the contents of the subdirectory where the examples reside. This allows you to experiment with the examples without affecting the integrity of the original files.

Header files

Table 2-1 describes header files required by all Client-Library applications.

Table 2-1: Required header files for Client-Library applications

File	Description
ctpublic.h	Required by all application source files that contain calls to Client-Library, it includes: • Definitions of symbolic constants used by Client-Library routines • Declarations for Client-Library routines
cspublic.h	The CS-Library header file, which includes: • Definitions of common client/server symbolic constants • Type definitions for common client/server structures • Declarations for CS-Library routines
bkpublic.h	Required in all application source files that make calls to bulk-copy routines
cstypes.h	Contains type definitions for Client-Library datatypes
sqlca.h	Contains a type definition for the SQLCA define structure

example.h file

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

```
** example.h

**

** This is the header file that goes with the Sybase

** Client-Library example programs.

**

**

*///* Sccsid %Z% %M% %I% %G% */

/*

** 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
*/
#ifdef vms
#include <stsdef.h>
#define EX_EXIT_SUCCEED     (STS$M_INHIB_MSG | STS$K_SUCCESS)
#define EX_EXIT_FAIL        (STS$M_INHIB_MSG | STS$K_ERROR)
#else
#define EX_EXIT_SUCCEED     0
#define EX_EXIT_FAIL        1
#endif /* vms */
/*
** Define global variables used in all sample programs
*/
#define EX_SERVER    NULL          /* use DSQUERY env var */
#define EX_USERNAME  "sa"
#define EX_PASSWORD  ""
/*
** Uncomment the following line to test the HA Failover feature.
*/
/* #define HAFAILOVER 1 */
/*
** For some platforms (e.g. windows 3.1), additional work needs to be done
** to insure that the output of some of the example programs can be displayed.
** QuickWin’s standard output screen buffer to unlimited size.
*/
#if QWIN
#define EX_SCREEN_INIT() _wsetscreenbuf(_fileno(stdout), _WINBUFINF)

#else /* QWIN */

#define EX_SCREEN_INIT()

#endif /* QWIN */
** This macro will insure that any setup is done for the platform.

**

** For windows, _wsetscreenbuf(_fileno(stdout), _WINBUFINT) will set
```

The changes made to EX_USERNAME and EX_PASSWORD include:

- EX_USERNAME is defined in *example.h* as "user.". Before you use the example programs, you must edit *example.h* and change "user" to your server login name.
- EX_PASSWORD is defined in *example.h* as "server_password." Before you use the example programs, you may want to edit *example.h* and change "server_password" to your server password.

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

  - *Method 1* – Change your server password to "server_password" for the duration of the time that you are running the examples. However, this method creates the possibility of a security breach. While your password is set to this published value, an unauthorized person could log in to the server as you. If this possibility presents a problem, choose one of the other methods of handling passwords for the example programs.
  - *Method 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 examples, edit the line so that it again reads "server_password."
  - *Method 3* – In the example programs, delete the ct_con_props code that sets the server password and substitute your own code to prompt users for their server passwords. (Because this code is platform-specific, Sybase does not supply it.)

Client-Library example programs

Example programs are included online with Client-Library to demonstrate typical uses for Client-Library routines. Some example programs use the sample databases supplied with Adaptive Server. Refer to your installation guide for information on installing the sample databases.

The example programs are C source files. The appropriate compiler must be installed on your platform if you plan to use the Client-Library example programs or build applications.

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 the following sections. Some of these programs require the unipubs database.

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.

Utility routines for the example programs

The *exutils.c* file contains utility routines that are used by all other Client-Library example 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 example source file.

firstapp.c example program

The *firstapp.c* example program is an introductory example that connects to the server, sends a select query, and prints the rows. This example is discussed in Chapter 1, "Getting Started With Client-Library," in the Open Client *Client-Library/C Programmer's Guide*.

exconfig.c example program

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

This example requires that you edit the default runtime configuration file, *\ini\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 *\ini\ocs.cfg* located in the Sybase installation directory (if necessary), and add the following 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*.

Bulk copy example program

The *blktxt.c* example program uses the bulk-copy routines to copy static data to a server table. In this program, three rows of data that are bound to program variables are sent to the server as a batch. The rows are sent again using blk_textxfer to send the text data.

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

Note This example requires SQL Server® version 11.1 or later.

Compute rows example program

The *compute.c* example program demonstrates processing compute results and performs the following:

- 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.

This 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 generates a compute row each time the value of type changes:

  ```
  compute sum(price) by type
  ```

- The second compute clause generates one compute row, which is the last to be returned:

  ```
  compute sum(price)
  ```

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

Note This example requires the pubs2 database and titles table.

Read-only cursor example program

The *csr_disp.c* example program demonstrates using a read-only cursor. First, it opens a cursor with a canned query. Then, it processes the results using the standard ct_results while loop. It binds the column values to program variables. Finally, 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 program, see the leading comments in the example source file.

Note This example requires SQL Server version 11.1 or later (or Adaptive Server), the pubs2 database, and the authors table.

Asynchronous example program

This example 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 example program contains two files:

- *ex_alib.c* – contains the source code of the library portion of the example. It is meant to be part of a library interface that supports asynchronous calls. This module provides a way to send a query to and retrieve the results from a server, all within one asynchronous operation.
- *ex_amain.c* – contains the source code of 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.

Note This example requires SQL Server version 11.1 or later (or Adaptive Server).

text and *image* example program

The *getsend.c* example program demonstrates how to retrieve and update text data from a table containing text and other datatypes. The process demonstrated can also be used for retrieving and updating image data. If connecting to an Open Server application, the Open Server application must be able to handle language commands intended for Adaptive Server.

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

Note This example requires SQL Server version 11.1 or later (or Adaptive Server), the pubs2 database, and the authors table.

Localization and internationalization example program

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.

Multithreaded example program

This example program contains two files, *multthrd.c* and *thrdfunc.c*, which demonstrate a multithreaded Client-Library application.

The example program comprises 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.

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

Note This example cannot run if your platform does not have a thread package supported by Client-Library. In addition, it requires SQL Server version 11.1 or later, or Adaptive Server.

RPC command example program

The remote procedure call (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 SQL Server version 11.1 or later.

Security service example program

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

For this example to execute, DCE or CyberSafe Kerberos must be installed and running on your machine. You must also connect to a server that supports network-based security, such as Security Guardian or the *secsrv.c* Open Server example 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 Microsoft Windows*.

Directory service example program

The directory service example program, *usedir.c,* queries 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 *sql.ini* 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 example program, see the leading comments in the example source file. For more information about directory services, refer to the Open Client and Open Server *Configuration Guide for Microsoft Windows*.

uni_blktxt.c example program

The *uni_blktxt.c* example 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 example program

The *uni_compute.c* example 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. First, it sends a canned query to the server using a language command. It processes the results using the standard ct_results while loop. Then, it binds the column values to program variables. Finally, it fetches and displays the rows in the standard ct_fetch while loop.

uni_csr_disp.c example program

The *uni_csr_disp.c* example program demonstrates using a read-only cursor. It is a modification of the *uni_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. It then 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 example program

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

uni_rpc.c **example program**

The RPC command example 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 example program, see the leading comments in the example source file.

usedir.c **example program**

This example 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 then displays a description of each entry found, and lets the user choose a server to connect to.

For more information about this example 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 **example program**

The *wide_compute.c* example program demonstrates processing compute results with wide tables and larger column sizes, implemented in Open Client and Open Server version 12.5. First, 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.

This 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 generates a compute row each time the value of type changes:

  ```
  compute sum(price) by type
  ```

- The second compute clause generates one compute row, which is the last to be returned:

  ```
  compute sum(price)
  ```

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 example program

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

This program uses a cursor to retrieve data from the publishers table 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 example program

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

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

CHAPTER 3

Open Client DB-Library/C Example Programs

Open Client DB-Library is a collection of routines you can use to write client applications. The example programs in this chapter demonstrate DB-Library functions. This chapter covers the following topics:

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.

How to use the example programs

The example programs demonstrate specific DB-Library/C functionality. These programs are designed as guides for application programmers, not as DB-Library/C training aids. Before you attempt to use the sample programs, read the descriptions at the top of each source file and examine the source code.

Note These simplified programs are not intended for use in a production environment. Programs written for a production environment need additional error-handling and special case-handling code.

Before you begin

Table 3-1 lists the steps you need to take to run all example programs for your platform. See the individual examples and the *README* file for additional requirements.

Table 3-1: Steps required to run DB-Library applications

Platform	Steps
All Windows platforms	• Set the DSQUERY environment variable to the server name (*server_name*) to which you want to connect. • Set the SYBASE environment variable to the path of the Sybase installation directory (if it has not already been set). • Use *makefile* to produce a sample executable named *example name*.

Location of the example programs

The location of the example programs is the *|sample\dblibrary* directory, located beneath the Sybase installation directory.

This directory contains:

- Online source code for the example programs
- Data files for the examples
- Header files, including *sybdbex.h*

Note Create a backup copy of the contents of the subdirectory where the examples reside. Then, you can freely experiment with the examples without affecting the integrity of the original files.

Header files

All DB-Library/C applications require these header files:

- *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. *sybfront.h* includes type definitions for datatypes that can be used in declaring program variables.
- *sybdb.h* – contains additional definitions and type definitions. Most of these are meant to be used only by 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.

sybdbex.h header file

All the example programs reference the example 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 the examples except the data conversion example program contain these lines:

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

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

The changes made to the EX_USERNAME, EX_PASSWORD, and LANGUAGE variables include:

- EX_USERNAME is defined in *sybdbex.h* as "user." Before you run the example programs, you must edit *sybdbex.h* and change "user" to your server login name.
- EX_PASSWORD is defined in *sybdbex.h* as "server_password." Before you run the example programs, you may want to edit *sybdbex.h* and change "server_password" to your server password.

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

  - *Method 1* – Change your server password to "server_password" while you are running the examples. However, this method creates the possibility of a security breach: While your password is set to this published value, an unauthorized person could log into the server as you. If this possibility presents a problem, choose one of the other methods.
  - *Method 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 examples, edit the line so that it again reads "server_password."
  - *Method 3* – In the example programs, delete the ct_con_props code that sets the server password and substitute your own code to prompt users for their server passwords. (Because this code is platform-specific, Sybase does not supply it.)
- LANGUAGE is defined in *sybdbex.h* as "us_english." If your server's language is not "us_english," you may want to edit *sybdbex.h* and change "us_english" to your server's language. The international languages routine example program, *exampl12.c*, is the only example that references LANGUAGE.

The DB-Library example programs

Example programs are included online with DB-Library to demonstrate typical uses for DB-Library routines. Some example programs use the sample databases supplied with Adaptive Server. Refer to your installation guide for information on installing the sample databases.

The example programs are C source files. You must install the appropriate compiler on your platform if you plan to use the DB-Library example programs or build applications.

Send queries, bind, and print results

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

Note Access to Adaptive Server is required.

Insert data into a new table

example2.c inserts data from a file into a newly created table, selects the server rows, and binds and prints the results.

Note Access to Adaptive Server is required. This example also requires a file named *datafile* (supplied) and create database permission in your login database.

Bind aggregate and compute results

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

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

Row buffering

example4.c demonstrates row buffering. It sends a query to Adaptive Server, buffers the returned rows, and allows you to examine them interactively.

Note Access to Adaptive Server is required.

Data conversion

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

Browse mode updates

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

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

Note Access to Adaptive Server is required.

Browse mode and ad hoc queries

example7.c 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 example demonstrates how an application can determine which columns resulting from ad hoc queries can be updated using browse-mode techniques.

This example 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 you selected can be browsed.

Note Access to Adaptive Server is required.

Making an RPC call

example8.c sends a remote procedure call (RPC), prints the result rows from the call, and prints the parameters and status returned by the remote procedure.

To use this example, you must have create 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.

Note Access to Adaptive Server is required.

Text and image routines

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

1. Inserts all data into the row except the text or image value.
2. Updates the row, setting the text or image value to NULL. This step is necessary, because a text or image value that is null will have a valid text pointer only if the null value was explicitly entered with the update statement.
3. Selects the row. You must specifically select the column that is to contain the text or image value. This step is necessary in order 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 command.
4. Calls dbtxtptr to retrieve the text pointer from DBPROCESS. dbtxtptr's *column* parameter is an integer that refers to the select operation performed in step 3. For example, if "text_column" is the name of the text column, the select statement reads:

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

dbtxtptr requires *column* to be passed as 3.

5 Calls dbtxtimestamp to retrieve the text timestamp from the DBPROCESS. dbtxtimestamp's *column* parameter refers to the select operation performed in step 3.

6 Writes 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.

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

Also, access to a Adaptive Server that contains the pubs2 database is required.

Inserting an image

exampl10.c prompts you for an author identification and the name of a file containing an image, reads the image from the file, and inserts a new row containing the author identification and the image into the pubs2 database table au_pix. For general information on inserting text and image values into a database table, see "Text and image routines" on page 41.

Note To run, *exampl10.c* requires access to Adaptive Server and the pubs2 database. The author identification must be of the form "*nnn-nn-nnnn*," where *n* is a numeric digit. Provided with the sample code, *imagefile* contains an image.

Retrieving an image

exampl11.c retrieves an image from the *au_pix* table in the pubs2 database. The author identification you enter determines which row the program selects. After retrieving the row, this example copies the image contained in the *pic* column to a file you specify.

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

- *exampl11.c* 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.
- You can also 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 an Adaptive Server that contains the pubs2 database is required.

International language routines

exampl12.c 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.

Bulk copy example program

The bulk-copy example 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.

Two-phase commit example program

This example, *twophase.c,* performs a simple update on two different servers. See the source code for the exact contents of the update. Once you have run the example, you can use isql on each of the servers to determine whether the update actually took place.

This example assumes that you have two Adaptive Servers running, named “SERVICE” and “PRACTICE,” each containing the pubs2 database. If your servers have different names, replace “SERVICE” and “PRACTICE” in the source code with the actual names of your servers.

Before running the example, make sure that your *interfaces* file includes appropriate entries for both servers. See the Open Client *DB-Library/C Reference Manual* and the Open Client and Open Server *Configuration Guide for Microsoft Windows* for information about the *interfaces* file.

If the “PRACTICE” server is on a different machine from the “SERVICE” server, the *interfaces* file on that machine must also contain an entry for the “SERVICE” query port. For details, see the Open Client *DB-Library/C Reference Manual*.

CHAPTER 4

Open Server Server-Library/C Example Programs

Open Server Server-Library is used for designing servers that take advantage of the features of the client/server architecture. These features include remote procedure calls (RPCs), multithreading, remote access to Adaptive Server, and others. The example programs discussed in this chapter demonstrate these features.

This chapter covers the following topics:

Topic	Page
How to use the example programs	46
Before you begin	46
Location and content	46
Tracing	47
Header files	48
The Server-Library example programs	48

You can create a complete, standalone server using Server-Library. Open Client-based applications and utilities, such as isql, take advantage of the features built into these servers.

How to use the example programs

The example programs demonstrate specific Server-Library/C functionality. These programs are designed as guides for application programmers, not as Server-Library/C training aids. Read the descriptions at the top of each source file and examine the source code before you attempt to use the example programs.

Note These simplified programs are not intended for use in a production environment. Programs written for a production environment need additional error-handling and special case-handling code.

Before you begin

Before you use the Open Server examples:

1. Set the SYBASE environment variable to the path of the Sybase release directory (if it has not already been set).
2. Set the DSLISTEN environment variable to your server's name.
3. Use *makefile* to produce a sample executable named *example_name*.

For detailed information regarding configuration of your environment and variables, see the Open Client and Open Server *Configuration Guide for Microsoft Windows*.

Location and content

Sample files that comes with Server-Library are located in the *\sample\srvlib subdirectory,* under the Sybase installation directory.

The *srvlib* subdirectory contains:

- Online source code for the example programs.
- *README* – a text file that has platform-specific and general notes about building, executing, and troubleshooting the examples.

- *makefile* – provided so that you can use it to build the examples. Use *makefile* as a starting point for your own Open Server applications.
- A SRV_CONNECT event handler.
- Error handlers.

Note Create a backup copy of the contents of the subdirectory where the examples reside. Then, you can freely experiment with the examples without affecting the integrity of the original files.

Tracing

Tracing provides detailed information about activities carried out by your application, depending on the options you select. The Open Server example programs support tracing, sending tracing output to the Open Server log file. To enable tracing, specify the following options on the command line when executing an example program:

```
example_name [normal_sample_options]
[-h] [-d] [-i] [-a] [-m] [-t] [-e] [-q] [-n]
```

Table 4-1 describes the type of tracing information provided by each option:

Table 4-1: Tracing options

Option	Description
-h	TDS header
-d	TDS data
-i	I/O
-a	Attention
-m	Message queue
-t	TDS token
-e	Event tracing
-q	Deferred event queue
-n	Net-Library tracing

Note -e and -q are mutually exclusive.

Header files

The following header files are required by all Open Server applications:

- *ospublic.h* – contains public Open Server structures, datatype definitions, define statements, and function prototypes
- *oserror.h* – contains Open Server error message numbers and text
- *oscompat.h* – contains mappings of old datatype definitions, datatypes, routines, constants, and function prototypes to new versions

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

The Server-Library example programs

This section contains information about the example programs that are included with Server-Library. The example programs demonstrate typical uses for Server-Library routines in C programs.

The example programs are C source files. You must have the appropriate compiler installed for your platform if you plan to use the Server-Library example programs or build applications. For more information on Sybase-supported compilers, see Chapter 1, “Building Open Client and Open Server Applications.”

Testing example programs

To test the example program, follow these steps:

1. Use isql to connect to *server_name*:

   ```
   isql -Uusername -Ppassword -S server_name
   ```

2. Enter the following query (or any SQL command):

   ```
   select * from sysservers
   ```

You may now execute any SQL commands.

A client program can stop an example by executing the stop_serv registered procedure from an Adaptive Server.

Open Server introduction

The *osintro.c* example program demonstrates the basic components that make up an Open Server application. *osintro.c* does not include a language handler and, therefore, cannot read commands sent by isql.

Gateway Open Server

ctosdemo.c is a gateway Open Server application. It uses Server-Library calls and Client-Library calls. First, 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. *ctosdemo.c* processes a variety of client commands, including:

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

In addition, *ctosdemo.c* 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.

Note Unlike other example programs included with Server-Library, *ctosdemo.c* attempts to be complete. It is provided as a coding template for use in a production environment. To terminate the *ctosdemo.c* program, press CTRL-C from the Command window. The command in the *README* file is incorrect.

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

srv_language event handler

lang.c 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*.

TDS passthrough mode

fullpass.c is an Open Server gateway application that demonstrates the Tabular Data Stream™ (TDS) passthrough mode. For more information, see the "Passthrough Mode" topics page in the Open Server *Server-Library/C Reference Manual*.

The event handler routine receives client requests using 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 using 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. Then, it 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.

Note This application requires access to Adaptive Server.

Registered procedures

regproc.c demonstrates the use of registered procedures in Open Server 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), which returns the version number of Open Server to the client
- *dbwait.c* – implemented with DB-Library, requests Open Server to notify the client when the registered procedure rp_version executes
- *ctwait.c* – implemented with Client-Library, requests Open Server to notify the client when the registered procedure rp_version executes

International languages and character sets

The *intlchar.c* example program demonstrates how Open Server handles international languages and character sets. It initializes values for the Open Server application native 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 and a SRV_CONNECT handler.

Multithreaded programming

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 using 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

multthrd.c installs a SRV_START handler, a SRV_LANGUAGE handler, a SRV_CONNECT handler, and callback handlers. A service thread logs all the language queries received by the Open Server application. The following occurs:

- 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.
- The client thread then waits (srv_sleep). When the service thread receives the message, it wakes up the client thread (srv_wakeup).
- The logger continuously loops waiting for messages. When it receives a message, it prints the contents of the query to a file and alerts 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.

Security services

The *secsrv.c* example program demonstrates how Open Server uses network-based security services.

The connection handler in this example 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 Microsoft Windows*.

CHAPTER 5 Open Client Embedded SQL/C

Embedded SQL™ is a superset of Transact-SQL® that allows you to embed Transact-SQL statements in application programs written in a language such as C. Embedded SQL includes all Transact-SQL statements, as well as certain extensions needed to use Transact-SQL in an application program.

This chapter covers the following topics:

Embedded SQL/C applications that have been precompiled with the cpre precompiler provide a simple method for retrieving, inserting, or modifying data stored in any Adaptive Server database. Refer to the Open Client *Embedded SQL/C Programmer's Manual* for information about Embedded SQL/C programming techniques.

Building an Embedded SQL/C executable

Following are basic steps to build an executable program from an Embedded SQL application:

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

The following sections describe each step.

Precompiling the application

Before you can compile your application, you must precompile the Embedded SQL/C code, as follows:

```
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]]
    [/Iinclude_directory]...
    [/Jlocale_for_charset]
    [/Ksyntax_level]
    [/L[listing_file_name]]
    [/Mlabelname=labelvalue]
    [/Nsql.ini]
    [/Otarget_file_name]
    [/P[password]]
    [/Sserver_name]
    [/Ttag_id]
    [/Uuser_id]
    [/Vversion_number]
    [/Zlocale_for_messages]
 program[.ext] [program[.ext]]...
```

Note Options can be flagged using either a slash (/) or a dash (-); therefore, cpre -I and cpre /I are equivalent.

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

You must enter *program*, the name of the Embedded SQL/C source file. The default extension for *program* is *.pc*. The cpre statement generates an output file with a *.c* extension.

Some of the options are switches that activate features of the precompiler, such as generating stored procedures. By default, these features are turned off. To turn them on, include 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).

Refer to Appendix B, "Precompiler Reference," for a detailed description of the cpreoptions.

Compiling and linking the application

Embedded SQL versions 11.1 and later are certified using the Microsoft C compiler on Windows platforms and the Borland C compiler on Windows NT. Refer to the *makefile* for the actual compile-and-link syntax. The *makefile* is located in the *\sample\esqlc* subdirectory beneath the Sybase release directory.

Link libraries

The following libraries are required on the link line:

- *libct* – Client-Library DLL
- *libcs* – CS-Library DLL
- *libtcl* – Transport Control Layer DLL
- *libcomn*– Internal Common library DLL
- *libintl* – Localization support library DLL

Loading stored procedures

Before running an Embedded SQL/C program, you must load the precompiler-generated stored procedures into Adaptive Server. To do this, you need to precompile the program with the -G option, which creates an isql script file. Then, use the isql -i option to load the created file.

For more information about isql, see Appendix A, “Utilities.”

The Embedded SQL/C example programs

Embedded SQL includes two online example programs that demonstrate typical Embedded SQL applications. Example programs are located in the *\sample\esqlc* subdirectory under the Sybase release directory. Included in this subdirectory are a *README* file and a *makefile*. This section gives a brief description of the example programs.

The sample programs demonstrate specific Embedded SQL/C functionality. 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 before attempting to use the sample programs. These simplified programs are not intended for use in a production environment: programs written for a production environment need additional error handling.

Before you begin

To run the Embedded SQL/C examples, set the SYBASE environment variable to the path of the Sybase installation directory (if it has not already been set).

Check the *sql.ini* file to ensure that there is an entry for the server name used. You can use dsedit to look at the *sql.ini* file. If you added servers to the *sql.ini* file, as described in the Open Client and Open Server *Configuration Guide for Microsoft Windows*, you can use ocscfg to test for connections to these servers.

To run the example programs, you must access an Adaptive Server that includes the pubs2 database. Refer to your installation guide for information on installing the pubs2 database.

Before you precompile the example programs, you must edit the example 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.

Header file

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

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


 #define USER            "sa"

 #define PASSWORD""
```

```
#define ERREXIT     -1
 #define STDEXIT     0
```

All the examples contain this line:

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

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

Using cursors for database query

example1 demonstrates how to use cursors in an interactive query program. The program:

- Displays a list of book types, from which the user selects one
- Displays all titles in the selected book type and 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

Displaying and editing rows of a table

example2 shows how to update 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.
- Sends the data to Adaptive Server after the user confirms the update.

CHAPTER 6

Open Client Embedded SQL/COBOL

Embedded SQL is a superset of Transact-SQL that allows you to embed Transact-SQL statements in application programs written in a language such as COBOL. Embedded SQL includes all Transact-SQL statements, as well as certain extensions needed to use Transact-SQL in an application program

This chapter covers the following topics:

Embedded SQL/COBOL programs that have been precompiled with the cobpre precompiler provide a simple method for retrieving, inserting, or modifying data stored in any SQL Server®/Adaptive Server® database.

Building an Embedded SQL/COBOL executable

To build an executable program from an Embedded SQL application:

1 Precompile the application.

2 Compile the COBOL source code generated by the precompiler.

3 Link your application, if required, to any necessary files and libraries.

4 Load any precompiler-generated stored procedures.

The following sections describe these steps.

Precompiling the application

The format of the statement to precompile an Embedded SQL source program is as follows:

```
cobpre    [/a] [/b] [/c] [/d] [/e] [/f] [/l]
      [/m] [/r] [/s] [/v] [/w] [/x] [/y]
     [/Ccompiler]
     [/Ddatabase_name]
     [/Ffips_level]
     [/G[isql_file_name]]
     [/Iinclude_directory]...
     [/Jlocale_for_charset]
     [/Ksyntax_level]
     [/L[listing_file_name]]
     [/Mlabelname=labelvalue]
     [/Nsql.ini]
     [/Otarget_file_name]
     [/P[password]]
     [/Sserver_name]
     [/Ttag_id]
     [/Uuser_id]
     [/Vversion_number]
     [/Zlocale_for_messages]
 program[.ext] [program[.ext]]...
```

Note Options can be flagged using either a slash (/) or a dash (-); therefore, cobpre -l and cobpre /l are equivalent.

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

program is the name of the Embedded SQL/COBOL source file. You must enter the name of the source file. The default extension for *program* is *.pco*. The cobpre option 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 “of” by default. To turn them “on,: include the option on the cobpre 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).

Refer to Appendix B, “Precompiler Reference,” for a detailed description of the cobpre options.

Compiling and linking the application

Embedded SQL version 11.1 and later has been certified using the MicroFocus net Express 3.1. Refer to the *makefile* located in the *\sample\esqlcob* directory under the Sybase installation directory for the actual compile-and-link syntax.

Link libraries

Some or all of the following libraries may be required on the link command line:

- *libcobct* – COBOL interface to Client-Library
- *libct* - Client-Library DLL
- *libcs* - CS-Library DLL
- *libtcl* – Transport Control Layer DLL
- *libcomn* – Internal common libraries DLL
- *libintl* – Localization support library DLL

Loading stored procedures

If you used the -G option when precompiling, you must load the precompiler-generated stored procedures into Adaptive Server. You can use the isql -i option to accomplish this task.

For more information on isql, see Appendix A, “Utilities.”

The Embedded SQL/COBOL example programs

Embedded SQL includes two example programs that demonstrate typical Embedded SQL applications. Example programs are located in the *\sample\esqlcob* subdirectory under the Sybase installation directory.

A *README* file in the same directory contains instructions for building and executing the examples and notes about using them. The *COBOL.pco* file defines a SQL Server/Adaptive Server login name and password. Update the login information in this file before compiling the examples.

The System 11 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 before attempting to use the sample programs. These simplified programs are not intended for use in a production environment. Programs written for a production environment need additional error handling.

General requirements

To run the Embedded SQL/COBOL examples, set the SYBASE environment variable to the path of the Sybase installation directory (if not already set).

You need to access a SQL Server/Adaptive Server on which the pubs2 sample database is installed. Refer to your installation guide for information on installing the pubs2 database.

Before you precompile the programs, replace the user name and password with values that are valid for your SQL Server/Adaptive Server. Comments in the programs show where you should make the changes.

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

Adding environment variable for MicroFocus COBOL

On UNIX platforms, add the COBOL home directory, *$COBDIR/coblib*, to the library path environment variable before running the Embedded SQL/COBOL sample programs.

Using cursors for database query

example1 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 and 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

Displaying and editing rows in a table

example2 demonstrates how to update a row using 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.
- Sends the data to SQL Server/Adaptive Server after the user confirms the update.

APPENDIX A Utilities

This appendix contains reference pages for the following utilities:

Utility	Description	Page
bcp	Bulk-copy utility, which copies a database table to or from an operating system file in a user-specified format.	65
defncopy	Definition copy utility, which 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.	81
isql	Interactive SQL parser, which connects to and queries an Adaptive Server or Open Server.	85

As an alternative to the bcp, defncopy, and isql utilities, Open Client provides SQL Advantage for Windows NT users. See the SQL Advantage *User's Guide* for more information.

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] [-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*]
[-P *password*]
[-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 owns 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
The name of the database table or view to copy.

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

datafile
The full path name of an operating system file. The path name can be from 1 to 255 characters in length.

-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*

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 datatype as the default. This option does not prompt for each field; it uses *char* as the default storage type, no prefixes, \t (tab) as the default field terminator, and \n (newline) as the default row terminator.

-e *errfile*

The full path 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. 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 prompts you to enter an explicit IDENTITY column value for each row. If the number of inserted rows exceeds the maximum possible IDENTITY column value, Adaptive Server returns an error.

By default, when you bulk copy data from a table with an IDENTITY column, bcp excludes all information about the column from the output file. If you specify the -E flag, bcp copies the existing IDENTITY column values into the output file.

-f *formatfile*
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 if 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*
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 (*sql.ini* on Windows platforms) located in the *ini* directory, which is below 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 disables character set conversion. 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. (See the *System Administration Guide* for more information about character sets and associated flags.

-K *keytab_file*
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*

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

-m *maxerrors*

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 not human-readable.

Warning! Do not use bcp in native format to perform data recovery, salvage, or to resolve an emergency situation. Do not use bcp in native format to transport data between different hardware platforms, different operating systems, or different major releases 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, and so on) the data will be unrecoverable.

-P *password*

Specifies an Adaptive Server password. If you do not specify -P *password*, bcp prompts for a password. If your password is NULL, place the -P flag at the end of the command line by itself.

-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.

In Japanese language environments, the -q flag translates Hankaku Katakana (half-width characters) into Zenkaku Katakana (full-width characters).

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

-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

-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. Enables client-side conversion.

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

-z *language*
: 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 afterward 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 which is 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 Microsoft Windows*.

Examples

Example 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.

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

Example 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 length of field pub_name [40]:
 Enter field terminator [none]:

 Enter the file storage type of field city [char]:
 Enter prefix length of field city [1]:
 Enter length of field city [20]:
 Enter field terminator [none]:

 Enter the file storage type of field state [char]:
 Enter prefix length of field state [1]:
 Enter length of field state [2]:
 Enter field terminator [none]:
```

Example 3 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.

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

Example 4 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 specified by the prompts. 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 length of field pub_name [40]:
```

```
Enter field terminator [none]:

Enter the file storage type of field city [char]:
Enter prefix length of field city [1]:
Enter length of field city [20]:
Enter field terminator [none]:

Enter the file storage type of field state [char]:
Enter prefix length of field state [1]:
Enter length of field state [2]:
Enter field terminator [none]:
```

Usage

- bcp provides a convenient and 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 into 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.

New features

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 rewrite 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.

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 in order to copy data out to a file, or in order 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 and after you have backed up your database with dump database.

- While the select into/bulkcopy option is on, you are not allowed to 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 have inserted unlogged data into your database, and you then perform a dump transaction before performing 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-1 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 dumpable.

Table A-1: Comparing fast and slow bcp

	select into/ bulkcopy on	**select into/ bulkcopy off**
Fast bcp (no indexes or triggers on target table)	OK dump transaction prohibited	bcp dump transaction prohibited
Slow bcp (one or more indexes or triggers)	OK dump transaction prohibited	OK 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 beforehand 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; recreate the indexes and triggers; and then dump the database. However, you need to allocate disk space for the construction of indexes and triggers—about 2.2 times the amount of 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 non NULL fields
  - The field terminator, which can be any character string
  - Scale and precision for numeric and decimal datatypes

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 terminators
  - Accept the default lengths

  Table A-2 shows the defaults and possible alternate responses:

Table A-2: bcp prompts, their 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 Adaptive Server datatype where implicit conversion is supported.
Prefix length	• 0 for fields defined with datatype (not storage type) (*char* and all fixed-length datatype) • 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.

Prompt	Default provided	Possible responses
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 dbconvert in the Open Client *DB-Library/C Reference Manual*.

Note Be careful when you copy data from different releases of Adaptive Server, because not all releases have the same datatypes.

- A prefix length is a 1-byte, 2-byte, 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 silently 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 newline (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 can not identify which tab represents the end of the string. Since bcp always looks for the first possible terminator, in this case it will 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. “...” indicates that the pattern repeats for each field. The field length is 8 for each column, and “string” represents the 6-character field each time.

Table A-3: 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-4: 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/or prefixes.
- When copying data into a table, bcp observes any defaults defined for columns and user-defined datatypes. However, bcp ignores rules in order 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 -q parameter is missing, or you mistyped the name(s).

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 {in filename dbname | out filename dbname
     [owner.]objectname [[owner.]objectname]...
     [-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.

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. Do not use objectname when copying definitions in.

-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*
Allows you to specify 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 (*sql.ini* for Windows platforms) located in the *ini* directory below 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. (See the *System Administration Guide* 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 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

  q – Enable out-of-sequence detection

  r – Enable data replay 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.

-z *language*
: 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 *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 Microsoft Windows*.

Examples

Example 1 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 -Usa -P -SMERCURY in new_proc stagedb
```

Example 2 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.

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

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 file names 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. 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:

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

- The -y *sybase_directory* option has been removed.

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*]
[-Q *option*]|
[-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, type:

```
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).

-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.

-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.

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

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

-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.

-Q *option*
Allows HA Failover.

-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. The default timeout for logging into isql is 60 seconds.

-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.

-Y

Tells the Adaptive Server to use chain transactions.

-z *language*

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 Microsoft Windows*.

Examples

Example 1 Executes the command.

```
isql
 Password:

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

Example 2 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.

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

Example 3 reset clears the query buffer. quit returns you to the operating system.

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

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 enables you to override any aliases you set up.
- 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.
- Case is significant for the isql flags.
- isql displays only 6 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 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, to comment out the go command use:

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

  instead of:

  ```
  /*
  ```

```
go
*/
```

New features

isql for System 11 is built with Client-Library. The isql user interface is unchanged except:

- The 5701 ("changed database") server message is no longer displayed after login or 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 from 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*:

- To exit from isql: quit or exit
- To clear the query buffer: reset
- To execute an operating system command:!! *command*

See also

sp_addlanguage, sp_addlogin, sp_configure, sp_defaultlanguage, sp_droplanguage, and sp_helplanguage in the Adaptive Server *Reference Manua*l.

APPENDIX B Precompiler Reference

The Embedded SQL precompilers, cpre and cobpre, share the same syntax and flag information. This section provides a reference summary that applies to both.

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

```
cpre|cobpre [/a] [/b] [/c] [/d] [/e] [/f] [/l] [/m]      [/p(cpre only)] [/s] [/r] [/v]
[/w]
        [/x] [/y]
    [/Ccompiler]
    [/Ddatabase_name]
    [/Ffips_level]
    [/G[isql_file_name]]
    [/Iinclude_directory]...
    [/Jlocale_for_charset]
    [/Ksyntax_level]
    [/L[listing_file_name]]
    [/Mlabelname=labelvalue]
    [/Nsql.ini]
    [/Otarget_file_name]
    [/P[password]]
    [/Sserver_name]
    [/Ttag_id]
    [/Uuser_id]
    [/Vversion_number]
    [/Zlocale_for_messages]
 program[.ext] [program[.ext]]...
```

Note Options can be flagged using either a slash (/) or a dash (-);therefore, cpre -l and cpre /l are equivalent.

Parameters

/a

Allows cursors to remain open across transactions. (See the *Adaptive Server Enterprise Reference Manual* for information about cursors and transactions. If you do not use this option, cursors behave as though set close on endtran on 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 or Embedded SQL/COBOL program.

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

- For the 11.1 and later 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 and later 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 debugging feature of Client-Library by generating calls to ct_debug.

This option is useful during application development but should be turned off for final application delivery. For this option to work properly, the application must be linked and run with the libraries and DLLs located in the */devlib* directory of the Sybase release directory.

/d

Turns off delimited identifiers (identifiers delimited by double quotes) and allows quoted strings in SQL statements to be treated as character literals.

/e

When processing an exec sql connect statement, directs Client Library to use the external configuration file to configure the connection. Also see the /x option and CS_CONFIG_BY_SERVERNAME property in the Open Client *Client-Library/C Reference Manual*.

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

/f

Turns on the FIPS flagger for ANSI FIPS compliance checking.

/l

Turns off generation of #line directives. This option works with ESQL/C only.

/m

Runs the application in Sybase auto-commit mode, which 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 cpre only

When this option is used, a separate command handle is generated for each SQL statement in the module that has input host variables, and sticky binds is enabled on each command handle. This option improves performance of repeatedly executed commands with input parameters at the cost of increased storage space usage and longer first executions 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).

/r

Disables repeatable reads. If you do not use this option, a set transaction isolation level 3 statement, which executes during connect statements, is generated. The default isolation level is 1.

/s

Includes static function declarations.

/v

Displays the precompiler version information only (without precompiling).

/w

Turns off display of warning messages.

/x

Uses external configuration files. See CS_EXTERNAL_CONFIG property described in the Open Client *Client-Library Reference Manual* and the INITIALIZE_APPLICATION statement described in the Open Client *Embedded/C SQL Reference Manual* and the Open Client *Embedded/COBOL SQL Reference Manual*.

/y
: Supports S_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:

- For COBOL, possible values include:

  "mf_byte" – Micro Focus COBOL with byte-aligned data (/C NOIBMCOMP).

  "mf_word" – Micro Focus COBOL with word-aligned data (/C IBMCOMP).

- For C, possible values include:

  "ANSI_C" – ANSI C compiler.

  "MSVC" – Microsoft Visual C compiler. The output of the "MSVC" precompiler will not generate strings longer than 1K.

/D*database_name*
: Specifies the name of the database to parse against. Use this option when you want to check SQL semantics 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.

/Ffips_level
: Checks for the specified conformance level. Currently, the precompiler can check for SQL89 or SQL92E.

/G[isql_file_name] (argument is optional)
: Generates stored procedures for appropriate SQL statements and saves them to a file for input to the database through isql. 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 *.isql* appended (or replacing any input file name extension).

  Also, see option -Ttag_id to specify tag IDs for stored procedures.

  If you do not use the /G option, no stored procedures are generated.

/Iinclude_directory

Specifies a directory where Embedded SQL will search for include files. You can specify this option any number of times. Embedded SQL searches the directories in command-line order. If you do not use this option, the default is the */include* directory of the Sybase release directory 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 the default, the precompiler looks for a locale name. CS-Library searches for the information in the following order:

- LC_ALL
- LANG

  If LC_ALL is defined, CS-Library uses its value as the locale name. If LC_ALL is not defined but LANG is defined, CS-Library uses its value as the locale name. If none of these locale values are defined, CS-Library uses a locale name of "default."

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

/K syntax_level

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

- NONE
- SYNTAX
- SEMANTIC

NONE is the default value. If you use either SYNTAX or SEMANTIC, you must also specify the /U, /P, /S, and /D options so that Embedded SQL can connect to your Adaptive Server.

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.

/Llisting_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.

/M
: Turns on security feature. Sets B1 secure labels.

/N sql.ini
: Specifies the configuration file name to the precompiler.

/O target_file_name
: Specifies the target or output file name. If you have multiple input files, you may not use this option (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 the extension *.cbl* appended (or replacing any input file name extension).

/P password](used with option /Uuser_id; argument is optional)
: Specifies an Adaptive Server password for SQL syntax checking at precompile time. /P without an argument, or with the keyword NULL as an argument, specifies a null (““) password. If you use option /Uuser_id without using /P, the precompiler prompts you to enter a password. Must be used with the /G flag.

/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. If DSQUERY is not set, then SYBASE is used as the name of the server.

/T tag_id (used with option /G)

Specifies a tag ID (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 ID *dbg* appended: *program_dbg;1*, *program_dbg;2*, and so on.

Programmers can use tag IDs 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 ID is added to the stored procedure name.

/U user_id

Specifies the Adaptive Server user ID.

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 are not using option /P[password], this option prompts you to enter a password.

Also, see /K, /P, /S, and /D options.

/V version_number

Specifies the Client-Library version number. 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_125 for Open Client and Open Server version 12.5).

/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 performs the following, in order:

1. Looks for a locale name. CS-Library searches for the information in the following order:

   - LC_ALL
   - LANG

   If LC_ALL is defined, CS-Library uses its value as the locale name. If LC_ALL is not defined but LANG is defined, CS-Library uses its value as the locale name. If none of these locale values are defined, CS-Library uses a locale name of “default.”

2. Looks up the locale name in the *locales* file to determine which language and character set are associated with it.

3. Loads localized messages and character set information appropriate to the language and character set determined in step 2.

program[.ext] [program[.ext]] . . .

Specifies the input file name(s) of the ESQL/COBOL source program. You can enter as many input file names as you wish. The file name format and length can be anything you want as long as it does not violate any rules. See “Comments” 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
```

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

```
cobpre|cpre /G /f program1.pco program2.pco
```

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

```
cobpre|cpre /a program1.pco program2.pco
```

4 Display the precompiler version information only:

```
cobpre|cpre /v
```

5 Run the precompiler with the highest level of SQL checking:

```
cobpre|cpre /K SEMANTIC /Uuser_id /Ppassword /Sserver_name \
     /Dpubs2 example1.[pc|pco]
```

Usage

- The cobpre|cpre command defaults are set up for ANSI standard behavior.
- The /a, /c, /f, /m, /r, and /V options affect only the connect statement. If your source file does not contain a connect statement, or if you use /e or /x , these options have no effect.
- Target file:
  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*, etc.
- Option format:
  Options will work with or without a space before the argument.
  For example, either of these formats will work:

  /Tdbg
  or
  /T dbg
- The precompiler can handle multiple input files. However, you may not use the option /O *target_file_name*, but must accept the default target file names (see "Target file" above). If you use option /G[isql_file_name], you cannot 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 cannot 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. These steps must be performed at the DOS command prompt.

1. Run the precompiler with options /c, /Ddatabase_name, /P[password], /Sserver_name, /K[SYNTAX| SEMANTIC], and /Uuser_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 /Ddatabase_name, /G[isql_file_name], and /Ttag_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 \
        /iisql_file_name
   ```

   Run tests on your program.

3. Run the precompiler with options /Ddatabase_name 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 \
   /iisql_file_name
   ```

   The final distribution program is ready to run.

How precompilers determine the names of their servers

You can connect with an Adaptive Server at precompile time, which allows you to do additional syntax checking at that time. The precompiler determines the name of its server in one of three ways:

- Using the -S option on the cpre or cobpre command line
- Setting the DSQUERY variable
- Using the default value, "SYBASE"

The -S option overrides the value set by DSQUERY.

Following is the syntax for choosing a server on the precompile command line:

```
cobpre|cpre -Usa -P -Sserver_name
```

As an alternative, you can leave the server name out of the connection call or statement, and *server_name* will take its value from the runtime value of the DSQUERY environment variable. If the application user has not set DSQUERY, the runtime value for the server name defaults to "SYBASE." See the Open Client and Open Server *Configuration Guide for Microsoft Windows* for more information on DSQUERY.

cobpre | cpre defaults

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

Table B-1: cobpre | cpre defaults

Option	Default if option not used
/C*compiler*	The mf_byte compiler for COBOL. ANSI-C for C.
/D*database_name*	The default database on Adaptive Server.
/F*fips_level*	(No FIPS flags available.)
/G[*isql_file_name*]	No stored procedures are generated.
/I*include_directory*	Default directory is the */include* directory of the Sybase release directory.
locale_for_charset	[platform-specific]
/K[syntax | semantic | none]	If neither syntax nor semantic is selected, the default setting is "None."
/L[*listing_file_name*]	No listing file is generated.
/M	(currently unavailable)
/Nsql.ini	The *sql.ini* file in the */ini* directory of the Sybase release directory.
/Otarget_file_name	The default target file name is the input file name with the extension *.cbl* or *.c* appended (or replacing any input file name extension).
/P[password]	You are not prompted for a password unless you use /U*user_id*.
/Sserver_name	The default Adaptive Server name is taken from the DSQUERY environment variable.
/Ttag_id	No tag IDs are added to the stored procedure names generated with /G.
/Uuser_id	None.
/Vversion_number	CS_VERSION_110 for version 11.1.x and later, CS_VERSION_125 for version 12.5.x and later
/Z*locale_for_messages*	[platform/environment specific]

Index