SYBASE®

Programmers Supplement

Open Client™ and Open Server™

15.0

[Microsoft Windows]

DOCUMENT ID: DC35455-01-1500-05

LAST REVISED: December 2008

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

Contents

About This Book

This book supplements the Open Client™ and Open Server™ reference manuals and programmers 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 2000 (x86) Service Pack 4 (32-bit)
- Windows XP (x86) Service Pack 2 (32-bit)
- Windows 2003 (x86) Service Pack 1 (32-bit)
- Windows 2003 (x64) (Service Pack 1)

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, and isql 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. Table 1 lists the products and their related documents:

Table 1: Product documentation list

Product	Related documentation
Client-Library™	Open Client *Client-Library/C Reference Manual* Open Client and Open Server *Common Libraries Reference Manual* Open Client *Client-Library/C Programmers Guide*
DB-Library™	Open Client *DB-Library/C Reference Manual*

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

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

See the Open Server and SDK *New Features* for Microsoft Windows, Linux, and UNIX, for descriptions of new features available for Open Server and the Software Developer's Kit. This document is revised to include new features as they become available.

Other sources of information

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

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

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

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

To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs.

2 Click Document Types under Technical Documents from the navigation bar on the left. Then, click Certification Report.

3 In the Certification Report filter, select Product, Platform, and Timeframe. Then, click Go.

4 Click a Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1 Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

Table 2 describes the syntax conventions:

Table 2: Syntax conventions

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

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Open Client and Open Server documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

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 Windows platforms. It also describes the requirements for building Windows executables using Sybase libraries.

It covers the following topics:

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 *%SYBASE%\%SYBASE_OCS%\dll* directory

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 sample programs or to build applications. Sybase has certified Microsoft's Visual C++ Version 6.0. 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 15.0 on Windows platforms is certified to work with the Open Server and Sybase Adaptive Server® (called "SQL Server" in versions prior to 11.5) products shown in Table 1-1:

Table 1-1: Open Client compatibility

Open Client 15.0 platform	Open Server 15.0	Open Server 12.5.1	Adaptive Server 15.0	Adaptive Server 12.5.3
Windows x86 (2000, 2003, XP)	x	x	x	x
Windows x64 (2003)	x	x	x	x

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

In addition, note these compatibility issues for Open Client:

- 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 15.0 on Windows platforms is certified to work with the Client-Library and Adaptive Server products shown in Table 1-2:

Table 1-2: Open Server compatibility

Open Server 15.0 platform	Client-Library 15.0	Client-Library 12.5.1	Adaptive Server 15.0	Adaptive Server 12.5.3
Windows x86 (2000, 2003, XP)	x	x	x	x
Windows x64 (2003)	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 *%SYBASE%\%SYBASE_OCS%\include* directory. This directory stores the header files when installation is complete.
LIB	Must contain the directory path that points to the *%SYBASE%\%SYBASE_OCS%\lib* directory. This directory stores the import library files once installation is complete.
PATH	Must contain the directory path that points to the *%SYBASE%\%SYBASE_OCS%\dll* directory. This directory stores the Sybase DLLs once installation is complete.

Header files

This section lists the header files that you need to include when compiling your Open Client and Open Server applications.

DB-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.
- *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* also includes type definitions for datatypes that can be used in program variable declaration.
- *syberror.h* – contains error severity values and should be included if the program refers to those values.

Client-Library header files

- *ctpublic.h* – required by all Client-Library applications. 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.

Server-Library header files

- *ospublic.h* – required by all Server-Library applications. 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 1-4 lists the import libraries you should include when compiling and linking your application:

Table 1-4: Open Client and Open Server import libraries

DB-Library import libraries	Client-Library import libraries	Server-Library import libraries
libsybdb.lib – DB-Library	*libsybct.lib* – Client-Library	*libsybsrv.lib* – Server-Library
	libsybcs.lib – CS-Library	*libsybct.lib* – Client-Library
	libsybblk.lib – Bulk-Library	*libsybcs.lib* – CS-Library
	You need to link with the Bulk-Library import library *libsybblk.lib* only if you make Bulk-Library calls.	*libsybblk.lib* – Bulk-Library
		You need to link with the Bulk-Library import library *libsybblk.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 %SYBASE%\%SYBASE_OCS%*dll* directory. Table 1-5 lists the DLLs that are supplied with Open Client and Open Server libraries:

Table 1-5: Open Client and Open Server DLLs

DB-Library DLLs	Client-Library DLLs	Server-Library DLLs
libsybdb.dll - DB-Library	*libsybct.dll* - Client-Library	*libsybct.dll* - Client-Library
libsybintl.dll - Localization support library	*libsybcs.dll* - CS-Library	*libsybcs.dll* - CS-Library
libsybtcl.dll - Transport control layer	*libsybintl.dll* - Localization support library	*libsybintl.dll* - Localization support library
libsybcomn.dll - Internal common library	*libsybtcl.dll* - Transport control layer	*libsybtcl.dll* - Transport control layer
libsybunic.dll - Unicode-Library	*libsybcomn.dll* - Internal common library	*libsybcomn.dll* - Internal common library
	libsybblk.dll – Bulk-Library	*libsybsrv.dll* – Server-Library
	libsybunic.dll - Unicode-Library	*libsybblk.dll* – Bulk-Library
		libsybunic.dll - Unicode-Library

Configuration requirements

For the sample 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 master 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 1-6 lists Open Client and Open Server properties with platform-specific default values:

Table 1-6: 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 *%SYBASE%\ini* directory 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
DB-Library	DBSETFILE	The path and name of the *sql.ini* file.	The *sql.ini* file in the *%SYBASE%\ini* directory 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 *Client-Library/C Reference Manual* and the Open Client *Client-Library/C Programmers Guide*.

ct_callback

Any Client-Library application routine that is registered with Client-Library using ct_callback must be declared as CS_PUBLIC. 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 *libsybct.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. ct_debug routine works only when you use the debug version of *libsybct.dll*.

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

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 *Client-Library/C Reference Manual*.

Example compile-and-link operations

A *makefile* for building Client-Library sample programs is provided in the *%SYBASE%\%SYBASE_OCS%\sample\ctlib* directory. An example fragment of *makefile* for a Windows Client-Library application for use by a Microsoft Visual C/C++ compiler, version 6.0 is as follows:

```
#####################################################################
# 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\libsybblk.lib
CTLIB =        $(SYBASEHOME)\$(SYBASE_OCS)\lib\libsybct.lib
CSLIB =      $(SYBASEHOME)\$(SYBASE_OCS)\lib\libsybcs.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)
```

```
... compile and link lines for each Client-Library sample program goes here ...

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

There are a few things to note in this example:

- 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
 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 the preemptive scheduling mode, set the SRV_PREEMPT option to "true" using the srv_config function.

Preemptive mode programming overview

In the preemptive scheduling mode, all threads are executable at the same time. Thread scheduling is handled by the Windows. Preemptive scheduling prevents a single thread from monopolizing the server. When running your application in the 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 can offer better performance for applications that do not share much data between threads

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);
```

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

A *makefile* for building Server-Library sample programs is provided in %SYBASE%\%SYBASE_OCS%\sample\srvlib directory. An example fragment of *makefile* for compiling and linking a Windows 32-bit application is as follows:

```
#############################################################################
#
# Microsoft makefile for building Sybase Open Server Samples for Windows
#
#############################################################################
MAKEFILE=MAKEFILE
!ifndef SYBASE
SYBASEHOME=c:\sybase
!else
SYBASEHOME=$(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\libsybcs.lib $(SYBASEHOME)\lib\libsybct.lib
$(SYBASEHOME)\lib\libsybsrv.lib
BLKLIB = $(SYBASEHOME)\lib\libsybblk.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)
... compile and link lines for each Client-Library sample program goes here ...
clean:

      -del *.obj
      -del *.map
      -del *.exe
      -del *.err
/*
```

Note that in this example:

- 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 Sample Programs

Open Client Client-Library is a collection of routines for use in writing client applications. Client-Library includes routines that send commands to a server and other routines that process the results of those commands. 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 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.

This chapter covers the following topics:

Using Client-Library sample programs

The sample 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 sample programs.

Note These sample 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 sample programs:

- Set the SYBASE environment variable to the path of the Sybase release directory (if it has not already been set).
- Set the SYBASE_OCS environment variable to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of 15.0 version of the Open Client and Open Server products.
- Set the DSQUERY environment variable to the server name (*server_name*) to which you want to connect.
- Use make in conjunction with the provided *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 sample programs

The Client-Library sample programs are located in *%SYBASE%\%SYBASE_OCS%\sample\ctlib* directory. This directory contains:

- Source code for the sample programs
- Data files for the sample programs
- The header file, *example.h*, for the sample programs

Note Create a backup copy of the contents of the directory where the sample programs reside. This allows you to experiment with the sample programs 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 sample 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.
*/
/*
** Define symbolic names, constants, and macros
*/
#define EX_MAXSTRINGLEN         255
#define EX_BUFSIZE              1024
#define EX_CTLIB_VERSION        CS_VERSION_150
#define EX_BLK_VERSION          BLK_VERSION_150
#define EX_ERROR_OUT            stderr
#define EX_BADVAL               (CS_INT)-1
#define EX_MAX_ARR              64

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

** Define global variables used in all sample programs
*/
#define EX_SERVER       NULL    /* use DSQUERY env var */
#define EX_USERNAME     "sa"
#define EX_PASSWORD     ""

/*
** Uncomment the following line to test the HA Failover feature.
*/
/* #define HAFAILOVER 1 */
#define EX_SCREEN_INIT()
```

The changes made to EX_USERNAME and EX_PASSWORD include:

- EX_USERNAME is defined in *example.h* as “sa.” Before you use the sample programs, you must edit *example.h* and change “sa” to your server login name.
- EX_PASSWORD is defined in *example.h* as null (“ ”) string. Before you use the sample programs, you may want to edit *example.h* and change to your server password.

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

- *Method 1* – Change your server password to null (“ ”) string 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 sample programs.
- *Method 2* – In *example.h*, change the null (“ ”) string 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. After finishing the example, edit the line so that it reads “server_password” again.
- *Method 3* – In the sample programs, delete the ct_con_props code that sets the server password and substitute your own code to prompt users for their server passwords. (As this code is platform-specific, Sybase does not supply it.)

Sample program summaries

Sample programs are included with Client-Library to demonstrate typical uses for Client-Library routines. Some sample programs use the sample databases supplied with Adaptive Server. Refer to the Adaptive Server Enterprise *Installation Guide* for information on installing the sample databases.

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

Utility routines for the sample programs

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

The *wide_util.c* file contains generic routines that are used by the wide_* sample programs. The routines are as follows:

- The init_db routine allocates the context and initializes the library. It also installs the callback routines and is called at the beginning of several sample programs.
- The cleanup_db routine closes the connection to the server and cleans up the context structure. This function is called at the end of the *wide_curupd.c* and *wide_dynamic.c* sample programs.
- The connect_db routine connects to the server, then sets the appropriate user name and password.
- The handle_returns routine processes the return result type.
- The fetch_n_print routine fetches the bound data into a host variable.

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

First sample program

The *firstapp.c* sample 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 Programmers Guide*.

Modified first sample program

The *uni_firstapp.c* sample program is a modification of the *firstapp.c* sample 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 Programmers Guide*.

Array bind sample program

The *arraybind.c* sample program demonstrates the use of array binding in conjunction with a CS_LANG_CMD initiated by ct_command. The sample program uses a hard-coded query of a hard-coded table in the pubs2 database. This query is defined by a language command using a select statement. The *arraybind.c* program then processes the results using the standard ct_results while loop. It binds column values to program arrays, then fetches and displays the rows in the standard ct_fetch loop.

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

Note This sample requires the pubs2 database.

Asynchronous sample program

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

The sample program 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 sample program, see the leading comments in the example source file and the *EX_AREAD.ME* file.

Note This sample program requires Adaptive Server version 11.1 or later.

Bulk copy sample program

The *blktxt.c* sample 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 sample program, see the leading comments in the example source file.

Compute rows sample program

The *compute.c* sample program demonstrates processing of 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 fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

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

This query returns both regular rows and compute rows. The compute rows are generated by the two `compute` clauses:

- The first `compute` clause 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 sample program requires the pubs2 database.

Directory service sample program

The directory service sample 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 sample 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.

External configuration sample program

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

This example requires that you edit the default runtime configuration file, *ocs.cfg*, located in *%SYBASE%\%SYBASE_OCS%\ini* directory. You can also use SYBOCS_CFG environment variable to point to the *ocs.cfg* file.

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, edit *ocs.cfg* file (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*.

Implicit read-only cursor sample program

The *csr_disp_implicit.c* sample program demonstrates using an implicit read-only cursor:

- It opens a cursor with a canned query.
- It processes the results using the standard ct_results while loop.
- It binds the column values to program variables.
- It fetches and displays the rows in the standard ct_fetch while loop.

The program flow is the same as the *csr_disp.c* sample program, with the only difference being the usage of the CS_IMPLICIT_CURSOR option instead of CS_READ_ONLY in the first ct_cursor call. Although, the generated output is the same as the *csr_disp.c* example, the use of CS_IMPLICIT_CURSOR potentially reduces network traffic at the network level.

When using this example, it is important to set the CS_CURSOR_ROWS option to a value greater than 1.

This is the canned query:

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

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

Note This example requires Adaptive Server version 12.5.1 or later and the pubs2 database.

Localization and internationalization sample program

The *i18n.c* sample 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 sample program

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

The sample 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 sample program, see the leading comments in the example source files.

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

Read-only cursor sample program

The *csr_disp.c* sample program demonstrates using a read-only cursor:

- It opens a cursor with a canned query.
- It processes the results using the standard ct_results while loop.
- It binds the column values to program variables.
- It fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

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

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

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

Read-only cursor modified sample program

The *uni_csr_disp.c* sample program demonstrates using a read-only cursor. It is a modification of the *csr_disp.c* sample program and requires the unipubs2 database. It performs the following:

- 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 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 instructions on installing the unipubs2 database, read the *README* file available in the *%SYBASE%\%SYBASE_OCS%\sample\ctlibrary* directory.

RPC command sample program

The remote procedure call (RPC) command sample program, *rpc.c*, sends an RPC command to a server and processes the results.

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

Modified RPC command sample program

The *uni_rpc.c* sample program 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 unipubs2 database.

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

For instructions on installing the unipubs2 database, read the *README* file available in the *%SYBASE%\%SYBASE_OCS%\sample\ctlibrary* directory.

Security service sample program

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

To execute this sample program, DCE or 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 sample program.

For more information about this sample 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.

Scrollable cursors sample program

The *csr_disp_scrollcurs.c* sample program uses a scrollable cursor to retrieve data from the authors table in the pubs2 database. It performs the following:

- It sends a canned query to the server to open a cursor.
- It processes the results using the standard ct_results while loop.
- It binds the column values to program variables.
- It fetches and displays the rows in the standard ct_scroll_fetch while loop.

This example uses a single prefetch buffer and regular program variables. Following is the canned query:

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

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

Note This example requires Adaptive Server version 15.0 or later, with scrollable cursor support and the pubs2 database.

Modified scrollable cursors sample program

The *csr_disp_scrollcurs2.c* sample program uses a scrollable cursor to retrieve data from the authors table in the pubs2 database. It performs the following:

- It sends a canned query to the server to open a cursor.
- It processes the results using the standard ct_results while loop.
- It binds the column values to program variables.
- It fetches the rows using ct_scroll_fetch and displays them.

This example uses a scrollable cursor with arrays as program variables and array binding. A single ct_scroll_fetch call displays results in an array.

This is the canned query:

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

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

Note This example requires Adaptive Server version 15.0 or later, with scrollable cursor support and the pubs2 database.

text and *image* sample program

The *getsend.c* sample 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 sample program, see the leading comments in the example source files.

Note This sample program requires Adaptive Server version 11.1 or later, the pubs2 database, and the authors table.

Two phase commit sample program

The *twophase.c* sample program performs a simple update on two different servers. Once you run the example, use isql on each server to determine whether the update took place.

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

unichar and *univarchar* bulk-copy sample program

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

unichar and *univarchar* compute sample program

The *uni_compute.c* sample program demonstrates processing of compute results. It is a modification of the *compute.c* sample program for the unichar and univarchar datatypes and requires the unipubs2 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.

For instructions on installing the unipubs2 database, read the *README* file available in the *%SYBASE%\%SYBASE_OCS%\sample\ctlibrary* directory.

Wide tables compute sample program

The *wide_compute.c* sample program demonstrates processing of compute results with wide tables and larger column sizes. It 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 fetches and displays the rows in the standard ct_fetch while loop.

Following is the canned query:

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

This query returns both regular rows and compute rows. The compute rows are generated by the two compute clauses:

- The first compute clause 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 tables cursor sample program

The *wide_curupd.c* sample 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)
go
```

Wide table dynamic data sample program

The *wide_dynamic.c* sample 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 table RPC command sample program

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

CHAPTER 3

Open Client DB-Library/C Sample Programs

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

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

This chapter covers the following topics:

Using DB-Library sample programs

The sample 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 sample programs for your platform. See the individual sample programs 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). • Set the SYBASE_OCS environment variable to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of 15.0 version of the Open Client and Open Server products. • Use make in conjunction with the provided *makefile* to produce a sample executable named *example name.*

Location of the sample programs

The sample programs are located in the *%SYBASE%\%SYBASE_OCS%\sample\dblib* directory.

This directory contains:

- Source code for the sample programs
- Data files for the sample programs
- Header files, including *sybdbex.h*

Note Create a backup copy of the contents of the directory where the sample programs reside. Then, you can experiment with the sample programs 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 sample 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 sample programs.
 **
 **
 */

 #define USER             "sa"
 #define PASSWORD         ""
 #define LANGUAGE         "us_english"
 #define SQLBUFLEN        255
 #define ERR_CH           stderr
 #define OUT_CH           stdout
extern void            error();

int CS_PUBLIC err_handler PROTOTYPE((
DBPROCESS   *dbproc,
int         severity,
```

```
int         dberr,
int         oserr,
char       *dberrstr,
char       *oserrstr));

int CS_PUBLIC msg_handler PROTOTYPE((
DBPROCESS   *dbproc,
DBINT       msgno,
int         msgstate,
int         severity,
char        *msgtext,
char        *srvname,
char        *procname,
int          line));
```

All the sample programs except the data conversion sample program contain these lines:

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

The changes made to the USER, PASSWORD, and LANGUAGE variables include:

- USER is defined in *sybdbex.h* as “sa.” Before you run the sample programs, you must edit *sybdbex.h* and change “sa” to your server login name.
- PASSWORD is defined in *sybdbex.h* as null (“ “) string. Before you run the sample programs, you may want to edit *sybdbex.h* and change to your server password.

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

  - *Method 1* – Change your server password to “” 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 “” 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 sample 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 sample program, *exampl12.c*, is the only example that references LANGUAGE.

Sample program summaries

Sample programs are included with DB-Library to demonstrate typical uses for DB-Library routines. Some sample programs use the sample databases supplied with Adaptive Server. Refer to the Adaptive Server Enterprise *Installation Guide* for information on installing the sample databases.

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

Send queries, bind, and print results sample program

The *example1.c* sample program 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 new table sample program

The *example2.c* sample program 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 sample program

The *example3.c* sample program 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 sample program

The *example4.c* sample program 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 sample program

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

Browse mode updates sample program

The *example6.c* sample program 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 sample program

The *example7.c* sample program 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.

RPC call sample program

The *example8.c* sample program 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 sample program

The *example9.c* sample program 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.

Insert image sample program

The *exampl10.c* sample program 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 sample program" on page 40.

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.

Retrieve image sample program

The *exampl11.c* sample program 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 sample program

The *exampl12.c* sample program 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 sample program

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

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

Two-phase commit sample program

This example, *twophase.c*, performs a simple update on two different servers. See the source code for the exact contents of the update. After you have run the 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 Sample Programs

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

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

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

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

This chapter covers the following topics:

Using Server-Library sample programs

The sample 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 sample 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 sample programs:

1 Set the SYBASE environment variable to the path of the Sybase release directory (if it has not already been set).

2 Set the SYBASE_OCS environment variable to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of 15.0 version of the Open Client and Open Server products.

3 Set the DSLISTEN environment variable to your server's name.

4 Use make in conjunction with the provided *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 come with Server-Library are located in the *%SYBASE%\%SYBASE_OCS%\sample\srvlib* directory.

The *srvlib* directory contains:

- Source code for the sample programs.
- *README* – a text file that has platform-specific and general notes about building, executing, and troubleshooting the sample programs.
- *makefile* – provided so that you can use it to build the sample programs. 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 directory where the sample programs reside. Then, you can experiment with the sample programs 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 sample 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 sample 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.

Sample program summaries

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

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

Testing sample programs

Before you run the sample program, do these steps:

1 Check if you can access to the Adaptive Server that is configured for remote access. To do this, log into the Adaptive Server and enter the following command:

```
execute sp_configure
```

If the Adaptive Server has already been configured for remote access, the config_value and run_value columns for the "remote access" option should be 1. If config_value is 0, enter the following command:

```
execute sp_configure 'remote access', 1
```

2 Make sure that an entry for your Open Server name exists in the *sql.ini* file or the Windows registry file. Use the dsedit utility to create an entry in the sql.ini file or the Windows registry file. For more information on dsedit, see Open Client and Open Server *Configuration Guide* for Microsoft Windows.

3 Make sure that your Open Server name exists in the Adaptive Server's sysservers table. To check this, log into Adaptive Server and enter the following command:

```
execute sp_helpserver
```

This command lists all the servers that are available in the Adaptive Server's sysservers table. If the name of your Open Server does not appear in this list, enter the following command:

```
execute sp_addserver your_open_server_name
```

4 Set these environment variables, if not already set:

Environment Variable	Value
SYBASE	Location of the Sybase home directory.
DSLISTEN	Name of the Open Server application as listed in the *sql.ini* file or directory service.

Environment Variable	Value
DSQUERY	Name of the Open Server as listed in the *sql.ini* file or directory service.

Open Server introduction sample program

The *osintro.c* sample 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 sample program

The *ctos.c* sample program 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. *ctos.c* processes a variety of client commands, including:

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

In addition, *ctos.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 sample programs included with Server-Library, *ctos.c* attempts to be complete. It is provided as a coding template for use in a production environment. To terminate the *ctos.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 sample program

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

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

TDS passthrough mode sample program

The *fullpass.c* sample program 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 sample program

The *regproc.c* sample program 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 sample program

The *intlchar.c* sample 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 sample program

The *multthrd.c* sample 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 sample program

The *secsrv.c* sample 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 lets you embed Transact-SQL statements in application programs written in languages like C. Embedded SQL includes all Transact-SQL statements, and the extensions needed to use Transact-SQL in an application program.

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

This chapter covers the following topics:

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
    [-Ccompiler]
    [-Ddatabase_name]
    [-Ffips_level]
    [-G[isql_file_name]]
    [-H]
    [-Iinclude_path_name]
    [-Jcharset_locale_name]
    [-Ksyntax_level]
    [-L[listing_file_name]]
    [-Ninterface_file_name]
    [-Otarget_file_name]
    [-Ppassword]
    [-Sserver_name]
    [-Ttag_id]
    [-Uuser_id]
    [-Vversion_number]
    [-Zlanguage_locale_name]
    [@options_file]...
   [-a] [-b] [-c] [-d] [-e] [-f] [-h] [-l] [-m] [-p] [-r] [-s] [-u] [-v] [-w] [-x] [-y]
   filename[.ext]
```

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

Compiling and linking the application

Embedded SQL versions 11.1 and later are certified using the Microsoft C compiler on Windows platforms. Refer to the *makefile* for the actual compile-and-link syntax. The *makefile* is located in the *%SYBASE%\%SYBASE_OCS%\sample\esqlc* directory.

Link libraries

The following libraries are required on the link line:

- *libsybct* – Client-Library DLL
- *libsybcs* – CS-Library DLL
- *libsybtcl* – Transport Control Layer DLL
- *libsybcomn*– Internal Common library DLL
- *libsybintl* – Localization support library DLL
- *libsybunic* – Unicode 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, "Utility Commands Reference."

Using Embedded SQL/C sample programs

Embedded SQL includes two sample programs that demonstrate typical Embedded SQL applications and are located in the *%SYBASE%\%SYBASE_OCS%\sample\esqlc* directory. Included in this directory are a *README* file and a *makefile*. This section gives a brief description of the sample 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.

Note 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 sample programs, set these environment variables:

- SYBASE environment variable to the path of the Sybase installation directory (if it has not already been set).
- SYBASE_OCS environment variable to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of 15.0 version of the Open Client and Open Server products.

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 sample programs, you must access an Adaptive Server that includes the pubs2 database. Refer to the Adaptive Server Enterprise *Installation Guide* for information on installing the pubs2 database.

Before you precompile the sample 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 sample programs reference the example header file, *sybsqlex.h*. The contents of *sybsqlex.h* are as follows:

```
/**************************************************
```

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


#define USER      "username"

#define PASSWORD  "password"


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

All the sample programs 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.

Example 1: Using cursors for database query

The *example1.cp* sample program demonstrates how to use regular, non-scrollable cursors in an interactive query program. The sample programs for scrollable cursors are *example4.pc*.and example5.pc. The programs:

- Display a list of book types, from which the user selects one
- Display all titles in the selected book type and prompts for a title ID
- Display detailed information about the selected title and continues prompting for title IDs
- Exit when Return is pressed at a prompt

Example 2: Displaying and editing rows of a table

The *example2.cp* sample program 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.

ExampleHA: Using cursors for database query with HA-Failover

The *exampleHA.cp* sample program shows how Embedded SQL/C code can be used with High Availability (HA) Failover capability. The program is similar to *example1.cp*, with the addition of failover processing. Error handlers are used to detect and handle failover.

Uni_example1: Using cursors for database query with unichar/univarchar support

The *uni_example1.cp* sample program shows how cursors can be used to guide an interactive query of the titles table. The program is similar to *example1.cp*, with the addition of displaying unichar/univarchar columns. The program:

- Binds the character datatype to the unichar/univarchar column.
- Accesses unichar/univarchar data from the server, and displays in the character format of the client's character set.

Uni_example2: Displaying and editing rows of a table with unichar/univarchar support

The *uni_example2.cp* sample program shows how cursors can be used to display and edit rows of a table. The program is similar to *example2.cp*, with the addition of displaying unichar/univarchar columns. The program:

- Binds the character datatype to the unichar/univarchar column.
- Accesses unichar/univarchar data from the server, and displays in the character format of the client's character set.

CHAPTER 6

Open Client Embedded SQL/COBOL

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

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

This chapter covers the following topics:

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
    [-C compiler]
    [-D database_name]
    [-F fips_level]
    [-G[isql_file_name]]
    [-I include_path_name]
    [-J charset_locale_name]
    [-K syntax_level]
    [-L[listing_file_name]]
    [-N interface_file_name]
    [-O target_file_name]
    [-P password]
    [-S server_name]
    [-T tag_id]
    [-U user_id]
    [-V version_number]
    [-Z language_locale_name]
    [@ options_file]
    [-a] [-b] [-c] [-d] [-e] [-f] [-l] [-m] [-r] [-s] [-u] [-v] [-w] [-x] [-y]
    filename[.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 "off" 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 has been certified using the Micro Focus Net Express 4.0. Refer to the *makefile* located in the *%SYBASE%\%SYBASE_OCS%\sample\esqlcob* 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:

- *libsybcobct* – COBOL interface to Client-Library
- *libsybct* - Client-Library DLL
- *libsybcs* - CS-Library DLL
- *libsybtcl* – Transport Control Layer DLL
- *libsybcomn* – Internal common libraries DLL
- *libsybintl* – Localization support library DLL
- *libsybunic* – Unicode 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, “Utility Commands Reference.”

Using Embedded SQL/COBOL sample programs

Embedded SQL includes two sample programs that demonstrate typical Embedded SQL applications. Sample programs are located in the *%SYBASE%\%SYBASE_OCS%\sample\esqlcob* directory.

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

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.

Note 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 sample programs, set these environment variables:

- SYBASE environment variable to the path of the Sybase installation directory (if it has not already been set).
- SYBASE_OCS environment variable to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of 15.0 version of the Open Client and Open Server products.

You need to access an Adaptive Server on which the pubs2 sample database is installed. Refer to the Adaptive Server Enterprise *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 Adaptive Server. Comments in the programs show where you should make the changes.

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

Environment variables for Micro Focus COBOL

Set the environment variables listed in Table 6-1 before running the Embedded SQL/COBOL sample programs.

Table 6-1: Environment variables for COBOL compiler

Environment variable	Value
COBDIR	Absolute path of the COBOL compiler installation directory
COBLIB	*%COBDIR%\lib*
PATH	*%COBDIR%\bin;%COBDIR%\lib*
LIB	*%COBDIR%\lib;%LIB%*

Example 1: Using cursors for database query

The *example1.pco* sample program shows how to use regular, non-scrollable cursors in an interactive query program. The regular, non-scrollable cursor sample 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 when Return is pressed at a prompt

Sample programs for scrollable cursors are *example3.pco* and *example4.pco*. The scrollable cursor sample programs:

- Declare INSENSITIVE or SEMI_SENSITIVE scrollable cursors
- Display and print selections of book titles based upon hard-coded FETCHES with a direction and/or offset
- Exit when the program is finished

Example 2: Displaying and editing rows in a table

The *example2.pco* sample program 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 Adaptive Server after the user confirms the update.

APPENDIX A

Utility Commands Reference

This appendix contains information on bcp, defncopy, isql, instjava, and extrjava utility program commands.

Utility	Description	Page
bcp	Bulk-copy utility, which copies a database table to or from an operating system file in a user-specified format.	68
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.	92
isql	Interactive SQL parser, which connects to and queries an Adaptive Server or Open Server.	97
instjava	Install java utility, which installs a JAR from a client file into an Adaptive Server.	114
extrjava	Extract java utility, which copies a retained JAR and the classes it contains from an Adaptive Server into a client file.	118

bcp

Description

Copies a database table to or from an operating system file in a user-specified format. This utility is available in the *%SYBASE%\%SYBASE_OCS%\bin* directory.

Syntax

bcp [[*database_name*.]*owner*.]*table_name* [*:slice_number* | partition *partition_name*] {in | out} *datafile*
[-a *display_charset*]
[-A *packet_size*]
[-b *batch_size*]
[-c]
[-C]
[-d *discardfileprefix*]
[-e *errfile*]
[-E]
[-f *formatfile*]
[-F *firstrow*]
[-g *id_start_value*]
[-i *input_file*]
[-I *interfaces_file*]
[-J *client_character_set*]
[-K *keytab_file*]
[-L *lastrow*]
[-m *maxerrors*]
[-n]
[-N]
[-o *output file*]
[-P *password*]
[-Q]
[-r *row_terminator*]
[-R *remote_server_principal*]
[-S *server*]
[-t *field_terminator*]
[-T *text_or_image_size*]
[-U *username*]
[-v]
[-V [*security_options*]]
[-W]
[-X]
[-y *alternate_home_directory*]
[-Y]
[-z *language*]
[-Z *security_mechanism*]
[--colpasswd [[[db_name.[owner].]table_name.]
column_name [*password*]]]
[--hide-vcc]
[--initstring "*TSQL_command*"]
[--keypasswd [[db_name.[owner].]*key_name* [*password*]]]
[--maxconn *maximum_connections*]

[--show-fi]
[--skiprows *nSkipRows*]

Parameters

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

owner
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 to copy. The table name cannot be a Transact-SQL reserved word.

slice_number
The number of the slice of the database table to copy.

partition *partition_name*
The name of the partition in Adaptive Server. For multiple partitions, use a comma-separated list of partition names.

in | out
The direction of the copy. in indicates a copy from a file into the database table; 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. For multiple datafiles, use a comma-separated list of file names. For multiple datafiles and partitions, the number of datafiles and partitions must be the same.

-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. Use -a in conjunction with -J to specify 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.

The following error message appears if the character translation file(s) named with the -a parameter is missing, or you mistype the name(s):

```
Error in attempting to determine the size of a pair of
translation tables. : 'stat' utility failed.
```

-A *packet_size*

Specifies the network packet size to use for this bcp session. For example, the following example sets the packet size to 4096 bytes for this bcp session:

```
bcp pubs2..titles out table_out -A 4096
```

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

-b *batchsize*

The number of rows per batch of data copied. By default, bcp in copies *n* rows in one batch, where *n* is equal to the batch size. Batch size applies only when bulk copying in; it has no effect on bulk copying out. The smallest number bcp accepts for *batchsize* is 1.

Note Setting the batch size to 1 causes Adaptive Server to allocate one data page to one row copied in. This parameter only applies to fast bcp, and is only useful in locating corrupt rows of data. Use -b 1 with care – doing so causes a new page to be allocated for each row, and is a poor use of space.

-c

Performs the copy operation with char datatype as the default storage type for all columns in the data file. Use this format if you are sharing data between platforms. This parameter 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.

-C

Supports bulk copy of encrypted columns if Adaptive Server supports encrypted columns. -C enables the ciphertext option before initiating the bulk copy operation.

-d *discardfileprefix*

Logs the rejected rows into a dedicated discard file. The discard file has the same format as the host file and is created by appending the input file name to the discard file prefix supplied. You can correct the rows in this file and use the file to reload the corrected rows.

Sybase recommends that you use the -d *discardfileprefix* option in conjunction with the -e *errorfile* to help identify and diagnose the problem rows logged in the discard file.

-e *errfile*

The full path name of an error file where bcp stores *all* rows that bcp was unable to transfer from the file to the database. The error messages from the bcp program appear on your terminal and are also logged in the error file. bcp creates an error file only when you specify this parameter. If multiple sessions are used, the partition and filename information for the error is added to the error file.

Sybase recommends that you use the -e *errorfile* option in conjunction with the -d *discardfileprefix* to help identify and diagnose the problem rows logged in the discard file.

-E

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

By default, when you bulk copy data into a table with an IDENTITY column, bcp assigns each row a temporary IDENTITY column value of 0. This is effective only when copying data into a table. bcp reads the value of the ID column from the data file, but does not send it to the server. Instead, as bcp inserts each row into the table, the server assigns the row a unique, sequential IDENTITY column value, beginning with the value 1. If you specify the -E flag when copying data into a table, bcp reads the value from the data file and sends it to the server which inserts the value into the table. 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.

The -E parameter has no effect when you are bulk copying data out. Adaptive Server copies the ID column to the data file, unless you use the -N parameter.

You cannot use the -E and -g flags together.

-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 parameter only if you previously created a format file that you want to use now for a copy in or out. If you do not specify this parameter, bcp interactively queries you for format information.

-F *firstrow*
The number of the first row to copy from an input file (default is the first row). If multiple files are used, this option applies to each file.

Avoid using this parameter when performing heavy-duty, multi-process copying, as it causes bcp to generally spend more effort to run, and does not provide you with a faster process. Instead, use -F for single-process, ad-hoc copying.

Note --F cannot co-exist with --skiprows.

-g *id_start_value*
Specifies the value of the IDENTITY column to use as a starting point for copying data in.

You cannot use the -g and -E flags together.

-i *input_file*
Specifies the name of the input file. The Standard Input is used as the 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, bcp looks for the interfaces file, *sql.ini* located in *%SYBASE%\ini* directory.

-J *client_character_set*
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_character_set* requests that Adaptive Server convert to and from *client_character_set*, 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, which may not necessarily be the character set that the client is using. For more information about character sets and associated flags, see the Adaptive Server Enterprise *System Administration Guide.*

-K *keytab_file*
Specifies the path to the keytab file for authentication in DCE.

-L *lastrow*
The number of the last row to copy from an input file (default is the last row). If multiple files are used, this option applies to each file.

-m *maxerrors*
The maximum number of errors permitted before bcp aborts the copy. bcp discards each row that it cannot insert (due to a data conversion error, or an attempt to insert a null value into a column that does not allow them), each rejected row as one error. If you do not include this option, bcp uses a default value of 10.

When multiple partitions are used, this number will be used for every file.

-n
Performs the copy operation using native (operating system) formats. Specifying the -n parameter means bcp will not prompt for each field. Files in native data format are not human-readable.

Warning! Do not use bcp in native format for 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. Do not use field terminators (-t) or row terminators (-r) with bcp in native format. Results are unpredictable and data may get corrupted. 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, a table was truncated or dropped, hardware damage occurred, a database table was dropped, and so on), the data is unrecoverable.

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

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

-o *output_file*
Specifies the name of the output file. The Standard Output is used as the default.

-P *password*
Specifies an Adaptive Server password. If you do not specify -P *password*, bcp prompts for a password. You can leave out the -P flag if your password is NULL.

-Q
Provides backward compatibility with bcp version 10.0.4 for copying operations involving nullable columns.

-r *row_terminator*
Specifies the row terminator.

Warning! Do not use -t or -r parameters with bcp in native format. Results are unpredictable and data may get corrupted.

When specifying terminators from the command line with the -t or -r parameter, you must escape characters that have special significance to the command prompt shell. See the examples for bcp for more information. Either place a backslash in front of the special character or enclose it in quotes. This is not necessary when bcp prompts you (interactive mode).

-R *remote_server_principal*
Specifies the principal name for the server as defined to the security mechanism. By default, a server's principal name matches the server's network name (which is specified with the -S parameter or the DSQUERY environment variable). Use the -R parameter 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 specified by the DSQUERY environment variable.

-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
Displays the current version of bcp and a copyright message and returns to the operating system.

-V *security_options*

Specifies network-based user authentication. With this parameter, 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 parameter; any password supplied with the -P parameter 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
- d – Enable credential delegation and forward the client credentials to the gateway application
- 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

-W

Specifies that if the server to which bcp is attempting to connect supports neither normal password encryption nor extended password encryption, plain text password retries are disabled. If this option is used, the CS_SEC_NON_ENCRYPTION_RETRY connection property will be set to CS_FALSE, and plain text (unencrypted) passwords will not be used in retrying the connection.

Note The -W option and the CS_SEC_NON_ENCRYPTION_RETRY property are ignored in this release.

-X

Specifies that, in this connection to the server, the application initiates 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.

This option can result in normal or extended password encryption, depending on connection property settings at the server. If CS_SEC_ENCRYPTION is set to CS_TRUE, normal password encryption is used. If CS_SEC_EXTENDED_ENCRYPTION is set to CS_TRUE, extended password encryption is used. If both CS_SEC_ENCRYPTION and CS_SEC_EXTENDED_ENCRYPTION are set to CS_TRUE, extended password encryption is used as the first preference.

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

-y *alternate_home_directory*

Sets an alternate Sybase home directory.

-Y

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

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

-z *language*

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.

You can add languages to an Adaptive Server during installation or afterwards, using either the langinst utility or the sp_addlanguage stored procedure.

The following error message appears if an incorrect or unrecognized language is named with -z parameter:

```
Unrecognized localization object. Using default value 'us_english'.
Starting copy ...
=> warning.
```

-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 *%SYBASE%\%SYBASE_OCS%\ini* directory. If no *security_mechanism* name is supplied, the default mechanism is used.

Note The CS_LIBTCL_CFG property specifies the name and path to an alternative *libtcl.cfg* file. For details about this property, see the Open Client and Open Server *Client Libraries Reference Manual*.

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.

--colpasswd *column_name password*
Sets passwords for encrypted columns by sending "set encryption passwd *password* for column *column_name*" to ASE. This does not automatically apply passwords to other encrypted columns, even if the second column is encrypted with the same key. The password must be supplied a second time to access the second column.

--hide-vcc
Instructs bcp not to copy virtual computed columns (VCC) either to or from a datafile. When you use this parameter in bcp OUT, the datafile does not contain data for VCC; in bcp IN, the data file may not contain data for a VCC.

If this option is used, Adaptive Server does not calculate or send virtual computed column data.

--initstring *"TSQL_command"*
Sends Transact-SQL commands to ASE before data is transferred.

Result sets issued by the initialization string are silently ignored, unless an error occurs. If ASE returns an error, bcp stops before data is transferred and displays an error message.

--keypasswd *key_name password*
Sets passwords for all columns accessed by a key by sending "set encryption passwd *password* for key *key_name*" to ASE.

--maxconn *maximum_connections*
The maximum number of parallel connections permitted for each bulk copy operation. For example, the following example sets the maximum number of parallel connections permitted for each operation to 2:

```
bcp --maxconn 2
```

If you do not include this parameter, bcp uses a default value of 10.

--show-fi

Instructs bcp to copy functional indexes, while using either `bcp IN` or `bcp OUT`. If this parameter is not specified, Adaptive Server generates the value for the functional index.

--skiprows *nSkipRows*

Instructs bcp to skip a specified number of rows before starting to copy from an input file. The valid range for --skiprows is between 0 and the actual number of rows in the input file. If you provide an invalid value, an error message displays.

Note --skiprows cannot co-exist with the -F option. Use of --skiprows with the -F option results in an error message.

Examples

Example 1 The -c parameter copies data out of the publishers table in character format (using char for all fields). The -t field_terminator parameter ends each field with a comma, and the -r row_terminator parameter ends each line with a Return. bcp prompts only for a password.

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

Example 2 The -C parameter copies data out of the publishers table (with encrypted columns) in cipher-text format, instead of plain text. 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 -C
Password:
Enter the file storage type of field col1 [int]:
Enter prefix length of field col1 [0]:
Enter field terminator [none]:
Enter the file storage type of field col2 [char]:
Enter prefix length of field col2 [0]:
Enter length of field col2 [10]:
Enter field terminator [none]:
Enter the file storage type of field col3 [char]:
Enter prefix length of field col3 [1]:
Enter field terminator [none]:
```

Example 3 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 4 Copies data out of partition p1 of table t1 to the *mypart.dat* file in the current directory.

```
bcp t1 partition p1 out mypart.dat
```

Example 5 Copies data back into Adaptive Server using the saved format file, *pub_form*:

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

Example 6 Copies a data file created with a character set used on a VT200 terminal into the pubs2..publishers table. The -z flag displays bcp messages in French:

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

Example 7 Specifies that Adaptive Server send 40K of text or image data using a packet size of 4096 bytes:

```
bcp pubs2..publishers out -T 40960 -A 4096
```

Example 8 Copies the *mypart.dat* file from the current directory, into table t1 of partition p1.

```
bcp t1 partition p1 in mypart.dat
```

Example 9 Copies partitions p1, p2 and p3 to files *a*, *b* and *c* respectively, in the *\work2\data* directory.

```
bcp t1 partition p1, p2, p3 out \work2\data\a,
\work2\data\b, \work2\data\c
```

Example 10 Copies files data.first, data.last and data.other into partitions *p1*, *p2* and *p3* respectively.

```
bcp t1 partition p1, p2, p3 in data.first, data.last,
data.other
```

Example 11 Disables replication when *titles.txt* data is transferred into the pubs2 titles table:

```
bcp pubs2..titles in titles.txt -- initstring "set
replication off"
```

Note Because the set replication off command in this example is limited to the current session in Adaptive Server, there is no need to explicitly reset the configuration option after bcp is finished.

Example 12 Sets the password to pwd1 for encrypted column col1:

```
bcp mydb..mytable out myfile –U uuu –P ppp –-colpasswd
db..tbl.col1 pwd1
```

Example 13 Sets a prompt to enter the password for encrypted column:

```
bcp mydb..mytable out myfile –U uuu –P ppp –-colpasswd
db..tbl.col1
Enter column db..tbl.col1's password: ***?
```

Example 14 Reads the password for encrypted column col1 from external OS file passwordfile:

```
bcp mydb..mytable out myfile –U uuu –P ppp –-colpasswd
db..tbl.col1 < passwordfile
```

Example 15 Sets password pwd1 for encryption key key1:

```
bcp mydb..mytable in myfile –U uuu –p ppp –-keypasswd
db..key1 pwd1
```

Example 16 Creates the discard file *reject_titlesfile.txt*:

```
bcp pubs2..titles in titlesfile.txt -d reject_
```

Example 17 For MIT Kerberos, requests credential delegation and forwards the client credentials to MY_GATEWAY:

```
bcp -Vd -SMY_GATEWAY
```

Example 18 bcp ignores the first two rows of the input file *titles.txt*, and starts to copy from the third row.

```
bcp pubs2..titles in titles.txt -U username -P password
--skiprows 2
```

Example 19 Sets an alternate Sybase home directory to *C:\work\NewSybase*:

```
bcp tempdb..T1 out T1.out -yC:\work\NewSybase -Uuser1
-Psecret -SMYSERVER
```

Usage

- You cannot use named pipes to copy files in or out.
- Using --hide-vcc improves performance, as Adaptive Server does not transfer and calculate data from virtual computed columns.
- Although you can use any Transact-SQL command with --initstring as an initialization string for bcp, you must reset possible permanent changes to the server configuration after running bcp. You can, for example, reset changes in a separate isql session.
- *slice_number* is included for backward compatibility with Adaptive Server 12.5.x and earlier, and can be used only with round-robin partitioned tables.
- You can specify either *slice_number* or *partition_name*, not both.
- You can specify multiple partition and data files. Separate each partition name or data file with commas.
- If you do not specify partition_name, bcp copies to the entire table.
- 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.

- The bcp utility does not insert any row that contains an entry exceeding the character length of the corresponding target table column. For example, bcp does not insert a row with a field of 300 bytes into a table with a character column length of 256 bytes. Instead, bcp reports a conversion error and skips the row. bcp does not insert truncated data into the table. The conversion error is as follows:

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

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

- Functional indexes can be copied in and out of the Adaptive Server database using the --show-fi parameter.

- The data from the Virtual computed columns (VCC) can be eliminated from copying in and out of the Adaptive Server database using the --hide-vcc parameter.

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.

Using the -d option

- Specifying the -d option applies only when bulk copying in; it is silently ignored when used in bulk copying out.

- If you use multiple input files, one discard file is created for every input file that has an erroneous row.

- If bcp reaches the maximum errors allowed and stops the operation, the bcp logs all the rows from the beginning of the batch until the failed row.

Note If the discard file option is specified, the batch size is automatically adjusted and the message `Warning!!! Batch size adjusted to the value` ***`newbatchsize`***`, for the optimization of the discard file feature.` is displayed, when:

- -b *batchsize* is specified but the batch or row size is too big to hold all the rows of the batch in memory.
- The -b *batchsize* option is not specified.

Copying tables with indexes or triggers

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

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

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

Table A-1: Comparing fast and slow bcp

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

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

Note The performance penalty for copying data into a table that has indexes or triggers in place can be severe. If you are copying in a very large number of rows, it may be faster to drop all the indexes and triggers 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) parameter, bcp prompts only for your password, unless you supplied it with the -P parameter. If you do not supply either the -n, -c or -f *formatfile* parameter, 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.
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 versions 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.
- Be sure that 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.

- Table A-3 and Table A-4 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).
- `Unable to open the discard-file` ***`discardfilename`***`.`
- `I/O error while writing the bcp` ***`discardfilename`***`.`
- `Unable to close the file` ***`discardfilename`***`. Data may not have been copied.`

Permissions

You must have an Adaptive Server account and the appropriate permissions on the database tables, as well as the operating system files to use in the transfer to use bcp.

- To copy data into a table, you must have insert permission on the table.
- To copy a table to an operating system file, you must have select permission on the following tables:
  - the table to copy
  - sysobjects
  - syscolumns
  - sysindexes

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. This utility is available in the *%SYBASE%\%SYBASE_OCS%\bin* directory.

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

Syntax

```
defncopy
    [-a display_charset]
    [-I interfaces_file]
    [-J [client_charset]]
    [-K keytab_file]
    [-P password]
    [-R remote_server_principal]
    [-S [server_name]]
    [-U user_name]
    [-v]
    [-V [security_options]]
    [-X]
    [-z language]
    [-Z security_mechanism]
    {in file_name database_name |  out file_name database_name
    [owner.]object_name [[owner.]object_name...] }
```

Parameters

-a *display_charset*

Runs defncopy from a terminal where the character set differs from that of the machine on which defncopy is running.Use -a in conjunction with -J to specify 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.

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 Adaptive Server Enterprise *System Administration Guide*.

-I *interfaces_file*
Specifies the name and location of the interfaces file to search when connecting to Adaptive Server. If you do not specify -I, defncopy looks for an interfaces file, *sql.ini* located in the *%SYBASE%\ini* directory.

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

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

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

Omitting -J sets the character set to a default for the platform. The default may not be the character set that the client is using. For more information about character sets and their associated flags, see the Adaptive Server Enterprise *System Administration Guide* for your platform.

-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 parameter. *Keytab* files can be created with the DCE dcecp utility. See your DCE documentation for more information.

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

-P *password*
Allows you to specify your password. If you do not specify -P, defncopy prompts for 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 parameter or the DSQUERY environment variable). Use the -R parameter when the server's principal name and network name are not the same.

-S *server_name*
Specifies the name of the Adaptive Server to connect to. If you specify -S with no argument, defncopy looks for a server named SYBASE. If you do not specify -S, defncopy uses the server specified by your DSQUERY environment variable.

-U *user_name*
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
Displays the version number and copyright message of defncopy and returns to the operating system.

-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 parameter; any password supplied with the -P parameter 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. defncopy (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which defncopy uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

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

-z *language*
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 langinst 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 *%SYBASE%\%SYBASE_OCS%ini* 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.

in | out
Specifies the direction of definition copy.

file_name
Specifies the name of the operating system file destination or source for the definition copy. The copy out overwrites any existing file.

database_name
Specifies the name of the database to copy the definitions to or from.

object_name
Specifies name(s) of database object(s) for defncopy to copy out. Do not use *object_name* when copying definitions in.

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

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 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 stated unambiguously. 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 of more than 100 characters that are placed before a create statement may cause defncopy to fail.

New features

defncopy 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.

Permissions

- 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 may not have select permission on the text column of the syscomments table if the System Security Officer has reset the allow select on syscomment.text column parameter with the system procedure sp_configure. This reset restricts select permissions to the object owner and the System Administrator. This restriction is required in order to run Adaptive Server in the *evaluated configuration*, as described in the Adaptive Server Enterprise installation and configuration documentation for your platform. In this case, the object owner or a System Administrator must execute defncopy to copy out definitions.

  Note If the text has been encrypted, it may be hidden from you even if you have all the required permissions. See "Verifying and Encrypting Source Text" in the Adaptive Server Enterprise *Transact-SQL User's Guide* for more information.

- 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.

isql

Description

Interactive SQL parser to Adaptive Server. This utility is available in the *%SYBASE%\%SYBASE_OCS%\bin* directory.

Syntax

isql [-b] [-e] [-F] [-n] [-p] [-v] [-X] [-Y] [-Q]
[-a *display_charset*]
[-A *packet_size*]
[-c *cmdend*]
[-D *database*]
[-E *editor*]
[-h *header*]
[-H *hostname*]
[-i *inputfile*]
[-I *interfaces_file*]
[-J *client_charset*]
[-K *keytab_file*]
[-l *login_timeout*]
[-m *errorlevel*]
[-o *outputfile*]
[-P *password*]
[-R *remote_server_principal*]
[-s *col_separator*]

[-S *server_name*]
[-t *timeout*]
[-U *username*]
[-V [*security_options*]]
[-w *column_width*]
[-W]
[-y *alternate_home_directory*]
[-z *localename*]
[-Z *security_mechanism*]
[--conceal [':?' | '*wildcard*']]
[--help]
[--retserverror]

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. Use -a in conjunction with -J to specify 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.

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 Adaptive Server Enterprise *System Administration Guide*.

-A *packet_size*

Specifies the network packet size to use for this isql session. For example, the following sets the packet size to 4096 bytes for this isql session:

```
isql -A 4096
```

To check your network packet size, enter:

```
select * from sysprocesses
```

The value is displayed under the *network_pktsz* heading.

packet_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, you can 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 in which the isql session begins.

-e
Echoes input.

-E *editor*
Specifies an editor other than your default editor (such as edit). To invoke it, enter its name as the first word of a line in isql.

-F
Enables the FIPS flagger. When you specify the -F parameter, the server returns a message when it encounters a non-standard SQL command. This option does not disable SQL extensions. Processing completes when you issue the non-ANSI SQL command.

-h *header*
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 *inputfile*
Specifies the name of the operating system file to use for input to isql . The file must contain command terminators ("go" by default).

- Specifying the parameter as follows is equivalent to < *inputfile*:

```
-i inputfile
```

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

-I *interfaces_file*
Specifies the name and location of the interfaces file to search when connecting to Adaptive Server. If you do not specify -I, isql looks for an interfaces file, *sql.ini* located in the *%SYBASE%\ini* directory.

-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. For more information about character sets and the associated flags, see the Adaptive Server Enterprise *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 isql must be logged in to DCE with the same user name as specified with the -U option.

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

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

-n
Removes numbering and the prompt symbol (>) from the echoed input lines in the output file when used in conjunction with -e.

-o *outputfile*
Specifies the name of an operating system file to store the output from isql. Specifying the parameter as -o *outputfile* is similar to > *outputfile*.

-p
Prints performance statistics.

-P *password*
Specifies your 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. If your password is NULL, use -P without any password.

-Q
Provides clients with failover (HA) property. See the Adaptive Server Enterprise *Using Sybase Failover in a High Availability System* for more information.

-R *remote_server_principal*
Specifies the principal name for the server as defined to the security mechanism. 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). Use -R when the server's principal name and network name are not the same.

-s *col_separator*
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.

The column separator appears at the beginning and the end of each column of each row.

-S *server_name*
Specifies the name of the Adaptive Server to connect to. isql looks this name up in the interfaces file. If you specify -S with no argument, isql looks for a server named SYBASE. If you do not specify -S, isql looks for the server specified by your DSQUERY environment variable.

-t *timeout*
Specifies the number of seconds before a SQL 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. Login names 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
- d – Enable credential delegation and forward the client credentials to the gateway application
- 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 and then exits.

-w *column_width*

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.

-W

Specifies that if the server to which isql is attempting to connect supports neither normal password encryption nor extended password encryption, plain text password retries are disabled. If this option is used, the CS_SEC_NON_ENCRYPTION_RETRY connection property will be set to CS_FALSE, and plain text (unencrypted) passwords will not be used in retrying the connection.

Note The -W option and the CS_SEC_NON_ENCRYPTION_RETRY property are ignored in this release.

-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.

This option can result in normal or extended password encryption, depending on connection property settings at the server. If CS_SEC_ENCRYPTION is set to CS_TRUE, normal password encryption is used. If CS_SEC_EXTENDED_ENCRYPTION is set to CS_TRUE, extended password encryption is used. If both CS_SEC_ENCRYPTION and CS_SEC_EXTENDED_ENCRYPTION are set to CS_TRUE, extended password encryption is used as the first preference.

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

-y *alternate_home_directory*
Sets an alternate Sybase home directory.

-Y
Tells the Adaptive Server to use chain transactions.

-z *localename*
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 afterward with the utility langinst or the sp_addlanguage stored procedure.

-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 *%SYBASE%\%SYBASE_OCS%\ini* 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.

--conceal [':?' | '*wildcard*']

Hides your input during an isql session. The --conceal option is useful when entering sensitive information, such as passwords.

wildcard, a 32-byte variable, specifies the character string that triggers isql to prompt you for input during an isql session. For every wildcard that isql reads, isql displays a prompt that accepts your input but does not echo the input to the screen. The default wildcard is :?.

Note --conceal is silently ignored in batch mode.

For information on how to use the wildcard in an isql session, see "Using prompt labels and double wildcards in an isql session" on page 112.

--help

Lists all available command-line parameters for the isql utility with a short description of the functionality of each parameter.

--retserverror

Forces isql to terminate and return a failure code when it encounters a server error of severity greater than 10. When isql encounters this type of abnormal termination, it writes the label "Msg" together with the actual ASE error number to stderr, and returns a value of "2" to the calling program. As before, isql prints the full server error message to stdout.

Examples

Example 1 Puts you in a text file where you can edit the query. When you write and save the file, you are returned to isql. The query appears; type "go" on a line by itself to execute it:

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

Example 2 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
```

Example 3 Creates column separators using the "#" character in the output in the pubs2 database for store ID 7896:

```
isql -Usa -P -s#
1> use pubs2
2> go
1> select * from sales where stor_id = "7896"
#stor_id#ord_num               #date                              #
#-------#----------------------#--------------------------------#
#7896   #124152                #              Aug 14 1986 12:00AM#
#7896   #234518                #              Feb 14 1991 12:00AM#

(2 rows affected)
```

Example 4 For MIT Kerberos, requests credential delegation and forwards the client credentials to MY_GATEWAY:

```
isql -Vd -SMY_GATEWAY
```

Example 5 When isql encounters a server error of severity 10 or greater, it returns a value of "2" to the command prompt, prints the full server error message to stdout, and writes the label "Msg" together with the actual ASE error number to stderr.

```
C:\>isql -Uguest -Pguestpwd -SmyASE
--retserverror 2> isql.stderr
1> select no_column from sysobjects
2> go
Msg 207, Level 16, State 4:
Server 'myASE', Line 1:
Invalid column name 'no_column'.


C:\>echo %ERRORLEVEL%
2
C:\>type isql.stderr
Msg     207
C:\>
```

Example 6 When you use the --help option, isql returns a brief description of syntax and usage for the isql utility consisting of a list of available arguments.

```
C:\>isql --help

usage: isql [option1] [option2] ... where [options] are...
-b                 Disables the display of the table headers output.
-e                 Echoes input.
-F                 Enables the FIPS flagger.
-p                 Prints performance statistics.
-n                 Removes numbering and the prompt symbol when used
```

```
                    with -e.
-v                  Prints the version number and copyright message.
-W                  Turn off extended password encryption on connection
                    retries.
-X                  Initiates the login connection to the server with
                    client-side password encryption.
-Y                  Tells the Adaptive Server to use chained transactions.
-Q                  Enables the HAFAILOVER property.
-a display_charset  Used in conjunction with -J to specify 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 packet_size      Specifies the network packet size to use for this isql
                    session.
-c cmdend           Changes the command terminator.
-D database         Selects the database in which the isql session begins.
-E editor           Specifies an editor other than the default editor vi.
-h header           Specifies the number of rows to print between column
                    headings.
-H hostname         Sets the client host name.
-i inputfile        Specifies the name of the operating system file to use
                    for input to isql.
-I interfaces_file  Specifies the name and location of the interfaces file.
-J client_charset   Specifies the character set to use on the client.
-K keytab_file      Specifies the path to the keytab file used for
                    authentication in DCE.
-l login_timeout    Specifies the number of seconds to wait for the server
                    to respond to a login attempt.
-m errorlevel       Customizes the error message display.
-M labelname labelvalue
                    Used for security labels. See CS_SEC_NEGOTIATE for more
                    details.
-o outputfile       Specifies the name of an operating system file to store
the output from isql.
-P password         Specifies your Adaptive Server password.
-R remote_server_principal
                   Specifies the principal name for the server as defined to
                   the security mechanism.
-s col_separator   Resets the column separator character, which is blank by
                   default.
-S server_name     Specifies the name of the Adaptive Server to which to
                   connect.
-t timeout         Specifies the number of seconds before a SQL command times
                   out.
-U username        Specifies a login name. Login names are case sensitive.
-V [security_options]
```

```
                    Specifies network-based user authentication. Valid
                    [security_options]:
                    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.
                    d - Requests credential delegation and forwards client
                        credentials.
-w column_width     Sets the screen width for output.
-y sybase_directory
                    Sets an alternate location for the Sybase home directory.
-z localename       Sets the official name of an alternate language to display
                    isql prompts and messages.
-Z security_mechanism
                    Specifies the name of a security mechanism to use on the
                    connection.
-x trusted.txt_file Specifies an alternate trusted.txt file location.
--retserverror      Forces isql to terminate and return a failure code when it
                    encounters a server error of severity greater than 10.
--conceal [wildcard]
                    Obfuscates input in an ISQL session. The optional wildcard
                    will be used as a prompt.
```

Example 7 Sets an alternate Sybase home directory to *C:\work\NewSybase*:

```
C:\>isql -yC:\work\NewSybase -Uuser1 -Psecret
-SMYSERVER
```

Example 8 Changes password without displaying the password entered. This example uses “old” and “new” as prompt labels:

```
C:\>isql -Uguest -Pguest -Smyase --conceal
1> sp_password
2> :? old
3> ,
4> :?:? new
5> go
old
new
Confirm new
Password correctly set.
(return status = 0)
```

Example 9 Changes password without displaying the password entered. This example uses the default wildcard as the prompt label:

```
C:\>isql -Uguest -Pguest -Smyase --conceal
1> sp_password
2> :?
3> ,
4> :?:?
5> go
:?
:?
Confirm :?
Password correctly set.
(return status = 0)
```

Example 10 Activates a role for the current user. This example uses a custom wildcard and the prompt labels “role” and “password:”

```
C:\>isql -UmyAccount --conceal '*'
Password:
1> set role
2> * role
3> with passwd
4> ** password
5> on
6> go
role
password
Confirm password
1>
```

Usage

- Following are the commands you can use at isql prompt:
  - To terminate a command:

    ```
    go
    ```

  - To clear the query buffer:

    ```
    reset
    ```

  - To execute an operating system command:

    ```
    !! command
    ```

  - To exit from isql:

    ```
    quit
    ```

    or

```
exit
```

- You must set the SYBASE environment variable to the location of the current version of the Adaptive Server before you can use isql.
- The 5701 ("changed database") server message is no longer displayed after login or issuing a use database command.
- Error message format differs from earlier versions of isql. If you have scripts that perform routines based on the values of these messages you may need to rewrite them.
- 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 the following:

  ```
  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, "send", the second value for -c, overrides ".", the first value:

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

  This enables you to override any aliases you set up.
- To call an editor on the current query buffer, enter its name as the first word on a line. Define your preferred callable editor by specifying it with the EDITOR environment variable. If EDITOR is undefined, the default is edit.

  For example, if the EDITOR environment variable is set to *emacs*, invoke it from isql using "*emacs*" as the first word on a line.
- Execute operating system commands by starting a line with "!!" followed by the command.
- To clear the existing query buffer, type reset on a line by itself. isql discards any pending input. You can also press Ctrl-C anywhere on a line to cancel the current query and return to the isql prompt.

- Read in an operating system file containing a query for execution by isql as follows:

  ```
  isql -Ualma -Ppassword < input_file
  ```

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

  ```
  isql -Ualma -Ppassword < input_file > output_file
  ```

- To redirect output from the isql command line to a file, use the ">" and ">>" operators. For example, to write the output of the select @@servername command to a file or overwrite that file if it already exists, enter the following command:

  ```
  select @@servername
  go > output_file
  ```

  To write the output of the select @@version command to a new file or append that file if it already exists, enter the following command:

  ```
  select @@version
  go >> output_file
  ```

- Use the "|" operator to pipe output to another command at the isql command line. For example, to pipe the output of the sp_who command to grep and return the lines that contain the string "sa," enter the following command:

  ```
  sp_who
  go | grep sa
  ```

- 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.

- When using isql interactively, read and display an operating system file into the command buffer with the following command:

  ```
  :R filename
  ```

- When using isql interactively, you can change the current database with the following command:

  ```
  use databasename
  ```

- You can include comments in a Transact-SQL statement submitted to Adaptive Server by isql. Open a comment with "/*". Close it with "*/" as shown in the following example:

  ```
  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:

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

  Do not use the following:

  ```
  /*
  go
  */
  ```

- isql defines the order of the date format as month, date, and year (mm dd yyy hh:mmAM (or PM)) regardless of the locale environment. To change this default order, user the convert function.

New features

isql 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* parameter has been removed.

Additional commands within *isql*:

Table A-5: isql session commands

Command	Description
reset	Clears the query buffer
quit or exit	Exits from isql
vi	Calls the editor
!! *command*	Executes an operating system command
:r *filename*	Reads an operating system file
:R *filename*	Reads and displays an operating system file
use *dbname*	Changes the current database to *dbname*
command > *filename*	Redirects output from the preceding *command* to the file *filename*. If *filename* already exists, its content is overwritten.
command >> *filename*	Redirects output from the preceding *command* to the file *filename*. If *filename* already exists, output from *command* is appended to *filename*.
command | *application*	Pipes the output of a Transact-SQL *command* to an external *application*.

Using prompt labels and double wildcards in an *isql* session

In an isql session, the default prompt label is either the default wildcard :? or the value of *wildcard*. You can customize the prompt label by providing a one-word character string with a maximum length of 80 characters, after a wildcard. If you specify a prompt label that is more than one word, the characters after the first word are ignored.

Double wildcards such as :?:? specify that isql needs to prompt you twice for the same input. The second prompt requests you to confirm your first input. If you use a double wildcard, the second prompt label starts with Confirm.

Note In an isql session, isql recognizes :? or the value of *wildcard* as wildcards only when these characters are placed at the beginning of an isql line.

See also

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

instjava

Description

Installs a JAR from a client file into an Adaptive Server. This utility is available in the *%SYBASE%\%SYBASE_OCS%\bin* directory.

Syntax

```
instjava
   -f file_name
   [ -new | -update ]
   [ -j jar_name ]
   [ -S server_name ]
   [ -U user_name ]
   [ -P password ]
   [ -D database_name ]
   [ -I interfaces_file ]
   [ -a display_charset ]
   [ -J client_charset ]
   [ -z language ]
   [ -t timeout ]
   [-v]
```

Parameters

-f *file_name*

The name of the source file containing the classes to be installed in the database.

-new | -update

Specifies whether the classes in the file already exist in the database.

If you specify the new parameter, you cannot install a class with the same name as an existing class.

If you specify the update parameter, you can install a class with the same name as an existing class, and the newly installed class replaces the existing class.

-j *jar_name*

The name of the JAR containing the classes to be installed in the database. Indicates that the JAR file should be saved in the database and associated with the classes it contains.

-S *server_name*

The name of the server.

-U *user_name*

An Adaptive Server login name. If you omit the -U flag and parameter, or if you specify the -U flag with no parameter, Adaptive Server uses the current user's operating system login name.

-P *password*

An Adaptive Server password. If you omit the -P flag and parameter, instjava prompts for a password. If you specify the -P flag with no password, the null password is used.

-D *database_name*

The name of the database in which to install the JAR. If you omit the -D flag, or if you specify the -D flag with no parameter, the user's default database is used.

-I *interfaces_file*

The name and location of the interfaces file to search when connecting to Adaptive Server. If you omit the -I flag and parameter, or if you specify the -I flag with no parameter, the interfaces file in the directory designated by your SYBASE environment variable is used.

-a *display_charset*

Allows you to use instjava from a machine where the character set differs that of the server. Use -a in conjunction with -J to specify 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.

-J *client_charset*

Specifies the character set to use on the client. instjava 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. Use this if the client and server use the same character set.

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

-z *language*

The name of an alternate language for displaying instjava prompts and messages. Without the -z flag, instjava uses the server's default language. You can add languages to an Adaptive Server during installation or afterward, using the langinstall utility or the sp_addlanguage stored procedure.

-t *timeout*

Specifies the number of seconds before a SQL command times out. If you do not specify a timeout, the command runs indefinitely. This affects commands issued from within instjava, not the connection time. The default timeout for logging into instjava is 60 seconds.

-v

Prints the version number and copyright message for instjava and then exits.

Examples

Example 1 Installs *addr.jar* and its classes, but does not retain the association between the JAR and classes:

```
instjava -f '\home\usera\jars\addr.jar' -new
```

Example 2 Reinstalls *addr.jar* and associates its classes with the employees JAR name:

```
instjava -f '\home\usera\jars\addr.jar' -update -j employees
```

Usage

- You must set the SYBASE environment variable to the location of the current version of Adaptive Server before you can use instjava.
- Refer to *Java in Adaptive Server Enterprise* for more information about how this utility is used when Java is enabled in the database.
- Any user can reference installed classes.
- The parameter flags -f, -j, -S, -U, -P, -D, and -I can be written with or without a space between the flag letter and the following parameter.

Adding new JARs

- If you use new with the -jar option and a JAR of that name already exists in the database, an exception is raised.
- If any classes of the same name as those in the source JAR already exist in the database, an exception is raised.

Updating JARs and classes

Warning! If you alter a class used as a column datatype by reinstalling a modified version of the class, you must make sure that the modified class can read and use existing objects (rows) in tables using that class as a datatype. Otherwise, you may be unable to access those objects without reinstalling the class.

- If you use -update with the -jar option:

    - All classes in the database associated with the target JAR are deleted from the database and the classes in the source JAR file installed in their place.
    - If a class in the source JAR file is already installed in the database but is not attached to a JAR, the class in the source JAR is installed in the database and the unattached class is deleted.
- If you use -update without the -jar option:
    - Classes in the source JAR file replace unattached classes of the same name.
    - Classes in the source JAR that do not correspond to an installed class are installed as unattached classes in the database.
- If you install a new JAR with a replacement for an installed class that is referenced by a SQLJ procedure or function, make sure that the newly installed class has a valid signature for the SQLJ routine. If the signature is invalid, an exception is raised when the SQLJ routine is invoked.

Locks

- When you execute instjava, an exclusive lock is placed on sysxtypes.
- If -jar is specified, an exclusive table lock is placed on sysjars.

Permissions: You need to be a System Administrator or Database Owner to use instjava.

Tables used: sysjars, sysxtypes

See also: **System procedures** sp_helpjava

Utilities extrjava

extrjava

Description

Copies a retained JAR and the classes it contains from an Adaptive Server into a client file. This utility is available in the *%SYBASE%\%SYBASE_OCS%\bin* directory.

Syntax

extrjava
-j *jar_name*
-f *file_name*
[-S *server_name*]
[-U *user_name*]
[-P *password*]
[-D *database_name*]
[-I *interfaces_file*]
[-a *display_charset*]
[-J *client_charset*]
[-z *language*]
[-t *timeout*]
[-v]

Parameters

-j *jar_name*

The name assigned to the retained JAR in the database that is the source of the transfer.

-f *file_name*

The name of the client file that is the target of the transfer.

-S *server_name*

The name of the server.

-U *user_name*

An Adaptive Server login name. If you omit the -U flag and parameter, or if you specify the -U flag with no parameter, Adaptive Server uses the current user's operating system login name.

-P *password*

An Adaptive Server password. If you omit the -P flag and parameter, extrjava prompts for a password. If you specify the -P flag with no password, the null password is used.

-D *database_name*

The name of the database in which to install the JAR. If you omit the -D flag, or if you specify the -D flag with no parameter, the user's default database is used.

-I *interfaces_file*
The name and location of the interfaces file to search when connecting to Adaptive Server. If you omit the -I flag and parameter, or if you specify the -I flag with no parameter, the interfaces file in the directory designated by your SYBASE environment variable is used.

-a *display_charset*
Allows you to use extrjava from a machine where the character set differs that of the server. Use -a in conjunction with -J to specify 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.

-J *client_charset*
Specifies the character set to use on the client. extrjava 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. Use this if the client and server use the same character set.

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

-z *language*
The name of an alternate language for displaying extrjava prompts and messages. Without the -z flag, extrjava uses the server's default language. You can add languages to an Adaptive Server during installation or afterward, using the langinstall utility or the sp_addlanguage stored procedure.

-t *timeout*
Specifies the number of seconds before a SQL command times out. If you do not specify a timeout, the command runs indefinitely. This affects commands issued from within extrjava, not the connection time. The default timeout for logging into extrjava is 60 seconds.

-v
Prints the version number and copyright message for extrjava and then exits.

Examples

Downloads the classes associated with the employees JAR to the client file *newaddr.jar*.

```
extrjava -j employees -f '\home\usera\jars\addr.jar' -new
```

Usage

- You must set the SYBASE environment variable to the location of the current version of Adaptive Server before you can use extrjava.
- If the target client file already exists, extrjava overwrites its contents.
- The parameter flags -f, -j, -S, -U, -P, -D, and -I can be written with or without a space between the flag letter and the following parameter.
- When you execute extrjava, an exclusive lock is placed on sysxtypes.
- If -jar is specified, an exclusive table lock is placed on sysjars.
- See the Adaptive Server Enterprise *Java in Adaptive Server Enterprise* for more information about how this utility is used when Java is enabled in the database.

Permissions

You need to be a System Administrator or Database Owner to use extrjava.

Tables used

sysjars, sysxtypes

See also

System procedures sp_helpjava

Utilities instjava

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.

cpre

Description

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

Syntax

cpre
[-C *compiler*]
[-D *database_name*]
[-F *fips_level*]
[-G [*isql_file_name*]]
[-H]
[-I *include_path_name*]...
[-J *charset_locale_name*]
[-K *syntax_level*]
[-L [*listing_file_name*]]
[-N *interface_file_name*]
[-O *target_file_name*]
[-P *password*]
[-S *server_name*]
[-T *tag_id*]
[-U *user_id*]
[-V *version_number*]
[-Z *language_locale_name*]
[@ *options_file*]
[-a] [-b] [-c] [-d] [-e] [-f] [-h] [-l] [-m] [-p] [-r] [-s] [-u] [-v] [-w] [-x] [-y]
filename[.ext]

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

Parameters

-C *compiler*
Specifies the target host language compiler values, such as:

- "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.

-F *fips_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.

-H
Generates code with HA-Failover capability.

-I *include_path_name*
Specifies a directory complete with the path name, 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 *charset_locale_name*

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.dat* file available in the *%SYBASE%\locales* directory 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.

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

If you have multiple input files or do not specify the argument, the default listing file name(s) will be the input file name(s) with the extension *.lis* appended (or replacing any input file name extension).

If you do not use this option, no listing file is generated.

-N *interface_file_name*
Specifies the interface file name, *sql.ini* 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)
Specifies an Adaptive Server password for SQL syntax checking at precompile time. Using -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_150 for Open Client and Open Server version 15.0).

-Z *language_locale_name*
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.dat* 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.

@options_file
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 *@options_file* contains names of the files to precompile, place the argument at the end of the command line.

-a
Allows cursors to remain open across 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. See the Adaptive Server Enterprise *Reference Manual* for information about cursors and transactions.

-b

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

The -b option differs in the 11.1 and 10.x versions of the Embedded SQL precompilers as follows:

- 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 *%SYBASE%\%SYBASE_OCS\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.

-h

Generates thread-safe code.

-l

Turns off generation of #line directives.

-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

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.

-u

Disables ANSI binds.

-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/C Reference Manual* and the INITIALIZE_APPLICATION statement described in the Open Client *Embedded SQL/C Programmers Guide*.

-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.

filename[.ext]

Specifies the input file name of the ESQL/C source program. The file name format and length can be anything you want as long as it does not violate any rules.

Examples

1 Run the precompiler (ANSI-compliant):

```
cpre program.pc
```

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

```
cpre -G -f program1.pc
```

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

```
cpre -a program1.pc
```

4 Display the precompiler version information only:

```
cpre -v
```

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

```
cpre -K SEMANTIC -Uuser_id -Ppassword -Sserver_name -Dpubs2 example1.pc
```

Usage

- The 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.
- 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.
- By default, *cpre* generate calls to ct_options that enable ANSI-style binding of indicator variables (CS_ANSI_BINDS). If indicator variables for nullable host variables (*columns*) are not available, Client-Library generates a fatal run-time error and aborts the application in use. You can avoid these issues by using -u with *cpre*. You may also disable ANSI binds by setting CS_ANSI_BINDS to cs_false in the *ocs.cfg* file.

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 -Ppassword -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 -Ppassword -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 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:

```
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.

cpre defaults

Table B-1 lists the options and defaults for the cpre and cobpre utilities:

Table B-1: cpre and cobpre 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_path_name*	Default directory is the *include* directory of the Sybase release directory.
-J *charset_locale_name*	[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.
-N *interface_file_name*	The *sql.ini* file in the *ini* directory of the Sybase release directory.
-O *target_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*.
-S server_name	The default Adaptive Server name is taken from the DSQUERY environment variable.
-T *tag_id*	No tag IDs are added to the stored procedure names generated with -G.
-U *user_id*	None.
-V *version_number*	CS_VERSION_125 for version 12.5.x and later CS_VERSION_150 for version 15.0 and later
-Z *language_locale_name*	[platform/environment specific]

cobpre

Description

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

Syntax

cobpre
[-C *compiler*]
[-D *database_name*]
[-F *fips_level*]
[-G [*isql_file_name*]]
[-I *include_path_name*]
[-J *charset_locale_name*]

```
[-K syntax_level]
[-L [listing_file_name]]
[-N interface_file_name]
[-O target_file_name]
[-P password]
[-S server_name]
[-T tag_id]
[-U user_id]
[-V version_number]
[-Z language_locale_name]
[@ options_file]
[-a] [-b] [-c] [-d] [-e] [-f] [-l] [-m] [-r] [-s] [-u] [-v] [-w] [-x] [-y]
filename[.ext]
```

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

Parameters

-C *compiler*

Specifies the target host language compiler values, such as:

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

-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.

-F *fips_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.

-I include_path_name

Specifies a directory complete with the path name, 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 *charset_locale_name*

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.dat* 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.

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

If you have multiple input files or do not specify the argument, the default listing file name(s) will be the input file name(s) with the extension *.lis* appended (or replacing any input file name extension).

If you do not use this option, no listing file is generated.

-M
Turns on security feature. Sets B1 secure labels.

-N *interface_file_name*
Specifies the configuration file name, *sql.ini* 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)
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. This 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_150 for Open Client and Open Server version 15.0).

-Z *language_locale_name*

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.dat* 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.

@options_file

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 *@options_file* contains names of the files to precompile, place the argument at the end of the command line.

-a

Allows cursors to remain open across 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. See the Adaptive Server Enterprise *Reference Manual* for information about cursors and transactions.

-b

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

The -b option differs in the 11.1 and 10.x versions of the Embedded SQL precompilers as follows:

- For the 11.1 and later versions of cobpre, 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 cobpre, 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 *%SYBASE%\%SYBASE_OCS\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.

-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.

-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.

-u
Disables ANSI binds.

-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/C Reference Manual* and the INITIALIZE_APPLICATION statement described in the Open Client *Embedded SQL/C Programmers Guide*.

-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.

filename[.ext]
Specifies the input file name of the ESQL/C source program. The file name format and length can be anything you want as long as it does not violate any rules.

Examples

1 Run the precompiler (ANSI-compliant):

```
cobpre program.pco
```

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

```
cobpre -G -f program1.pco
```

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

```
cobpre -a program1.pco
```

4 Display the precompiler version information only:

```
cobpre -v
```

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

```
cobpre -KSEMANTIC -Uuser_id -Ppassword -Sserver_name
-Dpubs2 example1.pco
```

Usage

- The cobpre| 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.

- By default, *cobpre* generate calls to ct_options that enable ANSI-style binding of indicator variables (CS_ANSI_BINDS). If indicator variables for nullable host variables (*columns*) are not available, Client-Library generates a fatal run-time error and aborts the application in use. You can avoid these issues by using -u with *cobpre*. You may also disable ANSI binds by setting CS_ANSI_BINDS to cs_false in the *ocs.cfg* file.

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 -Ppassword -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 -Ppassword -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 -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 defaults

See Table B-1 for a list of options and defaults for the for the cpre and cobpre utilities.

Index

D

E

F

G

H

I

L

M

O

P

R

S

T

U

W