SYBASE®
An SAP® Company

Programmers Supplement

Open Client™ and Open Server™

15.7

[Microsoft Windows]

DOCUMENT ID: DC35455-01-1570-02

LAST REVISED: June 2012

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

Contents

About This Book ..... **ix**

CHAPTER 1 **Building Open Client and Open Server Applications** **1**
- Open Client and Open Server requirements 1
  - C compilers 2
- Environment variables and header files 2
  - Header files 2
- Import libraries and dynamic link libraries (DLLs) 3
  - Import libraries 3
  - Dynamic link libraries (DLLs) 4
- Configuration requirements 5
- Platform-specific default values 6
- Client-Library programming issues 6
  - ct_callback 6
  - Using the debug DLLs 7
  - Multithreaded support 7
  - Example compile-and-link operations 7
- DB-Library programming issue 9
  - Compile-and-link example 9
- Server-Library programming issues 10
  - srv_callback 10
  - Scheduling modes 10
  - Preemptive mode programming 10
  - srv_sleep 11
  - srv_wakeup 11
  - Example of compile-and-link operations 12

CHAPTER 2 **Client-Library/C Sample Programs** **15**
- Using Client-Library sample programs 15
- Location of the sample programs 16
- Header files 17
  - example.h file 17
- Sample program summaries 19

Utility routines for the sample programs 19
firstapp.c sample program 19
uni_firstapp.c sample program 20
arraybind.c sample program 20
Asynchronous sample program 20
blktxt.c sample program 21
compute.c sample program 21
usedir.c sample program 22
exconfig.c sample program 22
csr_disp_implicit.c sample program 22
il8n.c sample program 23
Multithreaded sample program 23
batch_lang.c sample program 24
batch_dynamic.c sample program 24
csr_disp.c sample program 24
uni_csr_disp.c sample program 25
rpc.c command sample program 25
uni_rpc.c sample program 25
secct.c sample program 26
csr_disp_scrollcurs.c sample program 26
csr_disp_scrollcurs2.c sample program 27
getsend.c sample program 27
twophase.c sample program 28
uni_blktxt.c sample program 28
uni_compute.c sample program 28
wide_compute.c sample program 28
wide_curupd.c sample program 29
wide_dynamic.c sample program 30
wide_rpc.c sample program 30

CHAPTER 3 Open Client DB-Library/C Sample Programs 31
Using DB-Library sample programs 31
Location of the sample programs 32
Header files 32
sybdbex.h header file 33
Sample program summaries 35
example1.c sample program 35
example2.c sample program 35
example3.c sample program 35
example4.c sample program 36
example5.c sample program 36
example6.c sample program 36
example7.c sample program 36
example8.c sample program 37

example9.c sample program 37
exampl10.c sample program 38
exampl11.c sample program 38
exampl12.c sample program 39
bulkcopy.c sample program 39
twophase.c sample program 39

CHAPTER 4 Open Server Server-Library/C Sample Programs 41
Using Server-Library sample programs 42
Location and content 42
Tracing 43
Header files 44
Sample program summaries 44
Testing sample programs 44
osintro.c sample program 45
ctos.c sample program 45
dynlisten.c sample program 46
lang.c sample program 46
fullpass.c sample program 47
regproc.c sample program 47
halang.c sample program 48
intlchar.c sample program 48
mqueue.c sample program 48
multthrd.c sample program 48
paramreader.c sample program 49
redirect.c sample program 49
secsrv.c sample program 50
sendrpc.c sample program 50
timedsleep.c sample program 50
updtext.c sample program 51

CHAPTER 5 Open Client Embedded SQL/C 53
Building an Embedded SQL/C executable 53
Precompiling the application 53
Compiling and linking the application 54
Link libraries 55
Loading stored procedures 55
Using Embedded SQL/C sample programs 55
Header file 56
example1.cp sample program 57
example2.cp sample program 57
exampleHA.cp sample program 58
uni_example1.cp sample program 58

uni_example2.cp sample program 58

CHAPTER 6 Open Client Embedded SQL/COBOL 59
Building an Embedded SQL/COBOL executable 59
Precompiling the application 59
Compiling and linking the application 60
Link libraries 61
Loading stored procedures 61
Using Embedded SQL/COBOL sample programs 61
Environment variables for Micro Focus COBOL 62
example1.pco sample program 63
example2.pco sample program 63

APPENDIX A **Utility Commands Reference 65**
bcp 65
cpre 88
cobpre 99
defncopy 109
isql 114

APPENDIX B **Utility Messages 135**
bcp messages 135
Message 1: Memory allocation failure 135
Message 5: Unable to open input file 135
Message 6: Unable to open output file 136
Message 7: Bad arguments 136
Message 8: Invalid first row 136
Message 9: Invalid rows 136
Message 10: Invalid last row 137
Message 11: Invalid direction 137
Message 12: Invalid integer 137
Message 13: Duplicate flags 138
Message 14: Overriding arguments 138
Message 15: Invalid prefix length 138
Message 21: Retry 138
Message 23: Starting message 139
Message 24: N rows copied 139
Message 25: Total time 139
Message 26: File save 139
Message 27: Host file 139
Message 28: Invalid column type 140
Message 29: Invalid column type 140

Message 30: Average Time 140
Message 31: Copy failure 140
Message 32: Partial copy failure 141
Message 33: Invalid precision 141
Message 34: Invalid scale 141
Message 35: Unexpected result type 141
Message 36: Unexpected result 142
Message 37: Write error 142
Message 39: Invalid rows 142
Message 40: Row transfer error 143
Message 41: Invalid datatype 143
Message 42: Input read file error 143
Message 43: Error file write error 143
Message 44: Unable to open error file 144
Message 45: Unexpected end-of-file 144
Message 46: Negative-length prefix 144
Message 48: Cannot read specified number of rows 144
Message 49: Length prefix or terminator required 145
Message 50: Text/image data truncated 145
Message 51: Max errors exceeded 145
Message 52: Unable to open discard file 146
Message 53: Discard file write error 146
Message 54: Unable to close file 146
Message 55: Batch size adjusted 146
Message 56: Max rows reached 147
defncopy messages 147
Message 1: Memory allocation failure 147
Message 2: Insufficient read space 147
Message 3: Unable to open input file 148
Message 4: Unable to open output file 148
Message 5: Bad argument 148
Message 6: File not flushed 148
Message 7: Unexpected object definition 149
Message 8: Abend 149
Message 9: Invalid direction 149
Message 10: No object name 149
isql messages 150
Message 1: Memory allocation failure 150
Message 8: Database name length 150
Message 9: CS-Lib message callback routine installation 150
Message 10: CT-Lib initialization 150
Message 11: CT-Lib message callback routine installation 151
Message 12: Unsupported datatype 151
Message 13: Buffer overflow 151

Message 15: Invalid memory block size 151
Message 16: Invalid memory handle 152
Message 17: Internal memory allocation error 152
Message 18: Editor command too long 152
Message 19: Uninitialized application context 153
Message 20: Connection failure 153
Message 21: Unavailable command handle 153
Message 23: File position reset failure 153
Message 24: Command buffer not cleared 154
Message 25: Command not initiated 154
Message 26: Command handle not cleared 154
Message 28: Command argument too long 154
Message 29: Filename missing 155
Message 30: Prompt label too long 155
Message 31: Prompt input mismatch 155
Message 32: Missing quote 156
Message 33: Directory creation failure 156
Message 34: Unexpected argument type 156
Message 35: Unable to open history file 156
Message 36: Temporary file deletion failure 157

Index 159

About This Book

This book supplements the Open Client™ and Open Server™ reference manuals and programmers guide. It provides platform-specific information for creating, configuring, and troubleshooting applications using Open Client and Open Server products.

In this document, references to all Microsoft Windows platforms are referred to as "Windows," except where otherwise noted.

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.

How to use this book

This book contains these chapters:

- Chapter 1, "Building Open Client and Open Server Applications," provides information for building applications using Open Client and Open Server libraries.
- Chapter 2, "Client-Library/C Sample Programs," provides information on Client-Library sample programs, including their locations and what they do.
- Chapter 3, "Open Client DB-Library/C Sample Programs," provides information on DB-Library™ sample programs, including their locations and what they do.
- Chapter 4, "Open Server Server-Library/C Sample Programs," provides information on Server-Library sample programs, including their locations and what they do.
- Chapter 5, "Open Client Embedded SQL/C," provides information on Embedded SQL™/C sample programs, building an executable, and compiling and linking applications.

- Chapter 6, “Open Client Embedded SQL/COBOL,” provides information on Embedded SQL/Cobol sample programs, building an executable, and compiling and linking applications.
- Appendix A, “Utility Commands Reference,” contains references pages that detail the syntax, parameters, and qualifiers for the commands and utilities relevant to Open Client.
- Appendix B, “Utility Messages,” provides information about error, information, and warning messages for the bcp, defncopy, and isql utilities.

Related documents

You can see these books for more information:

- The *Open Server and SDK New Features for Windows, Linux, and UNIX*, which describes new features available for Open Server and the Software Developer’s Kit. This document is revised to include new features as they become available.
- The *Open Server Release Bulletin* for your platform contains important last-minute information about Open Server.
- The *Software Developer’s Kit Release Bulletin* for your platform contains important last-minute information about Open Client™ and SDK.
- The *jConnect™ for JDBC™ Release Bulletin* contains important last-minute information about jConnect.
- The *Open Client and Open Server Configuration Guide* for your platform contains information about configuring your system to run Open Client and Open Server.
- The *Open Client Client-Library/C Programmers Guide* contains information on how to design and implement Client-Library applications.
- The *Open Client Client-Library/C Reference Manual* contains reference information for Open Client Client-Library™.
- The *Open Server Server-Library/C Reference Manual* contains reference information for Open Server Server-Library.
- The *Open Client and Open Server Common Libraries Reference Manual* contains reference information for CS-Library, which is a collection of utility routines that are useful in both Client-Library and Server-Library applications.
- The *Open Server DB-Library/C Reference Manual* contains reference information for the C version of Open Client DB-Library™.

- The *Installation and Release Bulletin Sybase® SDK DB-Library Kerberos Authentication Option* contains information about installing and enabling the MIT Kerberos security mechanism to be used on DB-Library. DB-Library only supports network authentication and mutual authentication in the Kerberos security mechanism.
- The *Open Client and Open Server International Developers Guide* provides information about creating internationalized and localized applications.
- The *Open Client Embedded SQL™/C Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with C applications.
- The *Open Client Embedded SQL™/COBOL Programmers Guide* explains how to use Embedded SQL and the Embedded SQL precompiler with COBOL applications.
- The *jConnect for JDBC Programmers Reference* describes the jConnect for JDBC product and explains how to access data stored in relational database management systems.
- The *Adaptive Server® Enterprise ADO.NET Data Provider Users Guide* provides information on how to access data in Adaptive Server using any language supported by .NET, such as C#, Visual Basic .NET, C++ with managed extension, and J#.
- The *Adaptive Server Enterprise ODBC Driver by Sybase® Users Guide* for Microsoft Windows and UNIX, provides information on how to access data from Adaptive Server on Microsoft Windows and UNIX platforms, using the Open Database Connectivity (ODBC) Driver.
- The *Adaptive Server Enterprise OLE DB Provider by Sybase Users Guide for Microsoft Windows* provides information on how to access data from Adaptive Server on Microsoft Windows platforms, using the Adaptive Server OLE DB Provider.
- The *Adaptive Server Enterprise Database Driver for Perl Programmers Guide* provides information for Perl developers to connect to an Adaptive Server database and query or change information using a Perl script.
- The *Adaptive Server Enterprise extension module for PHP Programmers Guide* provides information for PHP developers to execute queries against an Adaptive Server database.

- The *Adaptive Server Enterprise extension module for Python Programmers Guide* provides information about Sybase-specific Python interface that can be used to execute queries against an Adaptive Server database.

Other sources of information

Use the Sybase Getting Started CD and the Sybase Product Documentation Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format. 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 Sybase Product Documentation Web site is accessible using a standard Web browser. In addition to product documentation, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Partner Certification Report.
3. In the Partner Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Partner Certification Report title to display the report.

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

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

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

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

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

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

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

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

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

3 Select a product.

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

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

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

Conventions

Table 1: Syntax conventions

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

Accessibility features

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

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

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

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

If you need help

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

CHAPTER 1

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.

Topic	Page
Open Client and Open Server requirements	1
Environment variables and header files	2
Import libraries and dynamic link libraries (DLLs)	3
Configuration requirements	5
Platform-specific default values	6
Client-Library programming issues	6
DB-Library programming issue	9
Server-Library programming issues	10

Open Client and Open Server requirements

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

- Have an ANSI-compliant C compiler installed
- Define the INCLUDE and LIB environment variables
- Set the PATH variable to include the *%SYBASE%\%SYBASE_OCS%\dll* directory

See the *Open Server and SDK New Features for Windows, Linux, and UNIX* for information on supported platforms.

C compilers

You must have an ANSI-compliant C compiler installed to use the sample programs or to build applications. Sybase has certified the Microsoft Visual C++ Version 6.0. and may certify other compilers. For a list of currently certified compilers, speak to your Sybase sales representative.

Compile and run an Open Client and Open Server program in the same way as any other C language program. For instructions about compiling and linking your application, see the documentation for your compiler.

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

For Windows supported platforms, see the *Open Server and SDK New Features for Windows, Linux, and UNIX*.

Environment variables and header files

For your applications to function properly, set a number of environment variables.

Table 1-1: 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

Depending on the product you are using, you may need to include one or more header files 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 1-2 lists the import libraries to include when compiling and linking your application:

Table 1-2: 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
	Link with the Bulk-Library import library *libsybblk.lib* only if you make Bulk-Library calls.	*libsybblk.lib* – Bulk-Library
		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 1-3 lists the DLLs that are supplied with Open Client and Open Server libraries:

Table 1-3: Open Client and Open Server DLLs

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

Configuration requirements

For the sample programs and your applications to run properly:

- 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 must 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 1-4 lists Open Client and Open Server properties with platform-specific default values:

Table 1-4: 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. For an example, ex_clientmsg_cb in *exutils.c* in the sample directory.

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 in the *debug* subdirectory of the Sybase *dll* directory and the non-debug version is 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 works only when you use the debug version of *libsybct.dll*.

See 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, see 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
```

In this example:

- Sybase libraries are made for Microsoft Windows x86 32-bit applications.
- SUBSYSTEM:CONSOLE indicates a console application.

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

DB-Library programming issue

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 example

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. For an example, cs_err_handler in *utils.c* in the sample directory.

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

In preemptive scheduling mode, all threads are executable at the same time. Thread scheduling is handled by 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 the Server-Library mutex facility rather than using the Windows-specific semaphore APIs.

Windows-specific preemptive programming uses 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 ensures that the sleeping thread is ready for srv_wakeup to be executed. This 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 Microsoft Windows x86 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
/*
```

In this example:

- Sybase libraries are made for Microsoft Windows x86 32-bit applications.
- SUBSYSTEM:CONSOLE indicates that this is a console application.

See 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 that is allocated in CS-Library.

Topic	Page
Using Client-Library sample programs	15
Location of the sample programs	16
Header files	17
Sample program summaries	19

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 require additional code to handle errors and special cases.

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 15.5 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 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 of the sample programs

The Client-Library sample programs are in *%SYBASE%\%SYBASE_OCS%\sample\ctlib*, which contains:

- Source code for the sample programs
- Data files for the sample programs
- The header files — *example.h*, *ctxact.h*, *exasync.h*, *exutils.h*, *thrdfuc.h*, *thrdutil.h*, and *wide_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, this file 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:

```
/*
** 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_CURRENT_VERSION
#define EX_BLK_VERSION          BLK_VERSION_155
#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()
```

Make these changes to EX_USERNAME and EX_PASSWORD:

- EX_USERNAME is defined in *example.h* as “sa.” Before you use the sample programs, edit *example.h* to 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 this value to your server password. You can:
  - Change your server password to null (“ ”) while you are running the examples. However, this creates the possibility of a security breach; an unauthorized person can 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.
  - 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.
  - 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 to demonstrate typical uses for Client-Library routines. Some sample programs use the Adaptive Server® sample databases. See the *Adaptive Server Enterprise Installation Guide* for information about installing the sample databases.

The sample programs are C source files. The appropriate compiler must be installed on your platform to use the sample programs or build applications.

Utility routines for the sample programs

The *exutils.c* file contains utility routines that are used by all other 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 these generic routines that are used by the wide_* sample programs:

- init_db – allocates the context and initializes the library. It also installs the callback routines and is called at the beginning of several sample programs.
- cleanup_db – 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.
- connect_db – connects to the server, then sets the appropriate user name and password.
- handle_returns – processes the return result type.
- fetch_n_print – fetches the bound data into a host variable.

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

firstapp.c sample program

firstapp.c 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*.

uni_firstapp.c sample program

uni_firstapp.c is a modification of *firstapp.c* 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*.

arraybind.c sample program

arraybind.c demonstrates the use of array binding 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. *arraybind.c* 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.

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 completion callbacks.

Asynchronous 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.

blktxt.c 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.c sample program

compute.c demonstrates processing of compute results:

- It sends a 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.

This is the 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.

---

usedir.c 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, see the *Open Client and Open Server Configuration Guide for Microsoft Windows*.

exconfig.c sample program

exconfig.c demonstrates how to externally configure Client-Library application properties.

This example requires you to edit the default runtime configuration file, *ocs.cfg*, located in *%SYBASE%\%SYBASE_OCS%\ini*. 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 "Using the Runtime Configuration File" in the *Open Client Client-Library/C Reference Manual*.

csr_disp_implicit.c sample program

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

- It opens a cursor with a 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 for *csr_disp.c*; the only difference is the use 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, set the CS_CURSOR_ROWS option to a value greater than 1.

This is the 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.

---

il8n.c sample program

i18n.c 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:

- *multthrd.c* – contains 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.

---

batch_lang.c sample program

The *batch_lang.c* sample program demonstrates how ct_send_params() can be used with a language statement. This sample uses ct_send_params() repeatedly to insert lines read from a file into a table. Since it uses the same location for the parameters for every line read, it does not need to call ct_param() or ct_setparam() in between calls to ct_send_params().

batch_dynamic.c sample program

The *batch_dynamic.c* sample program uses dynamic SQL and sends parameters to the server for which the data resides at different memory locations. Therefore, this sample also demonstrates how ct_setparam() can be used to rebind to different variables before calling ct_send_params() again.

csr_disp.c sample program

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

- It opens a cursor with a 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.

This is the 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 the pubs2 database.

uni_csr_disp.c sample program

uni_csr_disp.c is a modification of the *csr_disp.c* sample program and requires the unipubs2 database:

- It opens a cursor with a 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.

This is the query:

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

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

rpc.c 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.

uni_rpc.c sample program

uni_rpc.c 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 in *%SYBASE%\%SYBASE_OCS%\sample\ctlib*.

secct.c sample program

secct.c 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, see the *Open Client and Open Server Configuration Guide for Microsoft Windows*.

csr_disp_scrollcurs.c sample program

csr_disp_scrollcurs.c uses a scrollable cursor to retrieve data from the authors table in the pubs2 database:

- It sends a 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. This is 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.

---

csr_disp_scrollcurs2.c sample program

csr_disp_scrollcurs2.c uses a scrollable cursor to retrieve data from the authors table in the pubs2 database:

- It sends a 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 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.

getsend.c sample program

getsend.c demonstrates how to retrieve and update text data from a table containing text and other datatypes. The same process can be used to retrieve and update image data. If your application is 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 the pubs2 database, and the authors table.

twophase.c sample program

twophase.c 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.

uni_blktxt.c sample program

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

uni_compute.c sample program

uni_compute.c demonstrates processing of compute results for unichar and univarchar datatypes, and requires the unipubs2 database:

- It sends a 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 in the standard ct_fetch while loop.

For instructions on installing the unipubs2 database, see the *README* file in *%SYBASE%\%SYBASE_OCS%\sample\ctlib*.

wide_compute.c sample program

wide_compute.c demonstrates processing of compute results with wide tables and larger column sizes:

- It sends a 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.

This is the query:

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

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

- The first compute clause generates a compute row each time the value of type changes:

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

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

  ```
  compute sum(price)
  ```

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

Note This sample requires the pubs2 database.

---

wide_curupd.c sample program

wide_curupd.c uses a cursor to retrieve data from the publishers table in the pubs2 database. It retrieves data row by row and prompts the user for 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 command. 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_dynamic.c sample program

wide_dynamic.c 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. Each time you call the select, you pass only new values for the key value, which determines the row to be retrieved. The behavior is similar to passing input parameters to stored procedures. The program also uses cursors to retrieve rows one by one, which can be manipulated as required.

wide_rpc.c 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. Some functionality, such as Directory and Security services support, is not included with DB-Library. You must use Client-Library to take advantage of these 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.

Topic	Page
Using DB-Library sample programs	31
Location of the sample programs	32
Header files	32
Sample program summaries	35

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.

These are the steps required to run DB-Library applications:

- 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 15.5 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*.

See the individual sample programs and the *README* file for additional requirements.

Location of the sample programs

The sample programs are in *%SYBASE%\%SYBASE_OCS%\sample\dblib*, which 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:

```
/*
 ** 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);
```

Make these changes to the USER, PASSWORD, and LANGUAGE variables:

- USER is defined in *sybdbex.h* as "sa." Before you run the sample programs, edit *sybdbex.h* to change "sa" to your server login name.
- PASSWORD is defined in *sybdbex.h* as null (" "). You may want to edit *sybdbex.h* to change your server password.
    - Change your server password to "" while you are running the examples. However, this creates the possibility of a security breach; an unauthorized person can log in to the server as you. If this possibility presents a problem, choose one of the other methods.
    - 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."
    - 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* to 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 to demonstrate typical uses for DB-Library routines. Some sample programs use the Adaptive Server sample databases. See 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 to use the DB-Library sample programs or build applications.

example1.c sample program

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

Note This sample requires access to Adaptive Server.

example2.c sample program

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

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

example3.c sample program

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

Note This sample requires access to Adaptive Server that contains the pubs2 database.

example4.c sample program

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

Note This sample requires access to Adaptive Server.

example5.c sample program

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

example6.c sample program

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

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

Note This sample requires access to Adaptive Server.

example7.c sample program

example7.c uses browse-mode techniques to determine the source of result columns from ad hoc queries.

Determining the source of result columns is important, because a browse-mode application can update only 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. The results differ depending on whether the select query includes the keywords for browse and whether the table you selected can be browsed.

Note This sample requires access to Adaptive Server.

example8.c sample program

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

To use this example, you must 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 This sample requires access to Adaptive Server.

example9.c sample program

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

1 Inserts all data into the row except the text or image value.

2 Updates the row, setting the text or image value to NULL. This step is necessary, because a text or image value that is null has 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 to provide the application's DBPROCESS with correct text pointer and text timestamp information. The application should dispose of 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, this sample requires access to Adaptive Server that contains the pubs2 database.

---

exampl10.c sample program

exampl10.c prompts you for an author identification and the name of a file containing an image, reads the image from the file, and inserts a new row containing the author identification and the image into the pubs2 database table au_pix. See "example9.c sample program" on page 37.

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.

---

exampl11.c sample program

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

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

- *exampl11.c* selects the row containing the value and processes the row using dbnextrow. After dbnextrow is called, dbdata can be used to return a pointer to the returned image.
- You can also use dbreadtext 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 This sample requires access to Adaptive Server that contains the pubs2 database.

exampl12.c sample program

exampl12.c retrieves data from the pubs2 database and prints it using a us_english format.

Note This sample requires access to Adaptive Server that contains pubs2 database.

bulkcopy.c 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 This sample requires access to Adaptive Server. You must have create database and create table permission.

twophase.c sample program

twophase.c performs a simple update on two different servers. See the source code for the exact contents of the update. After you 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.

CHAPTER 4

Open Server Server-Library/C Sample Programs

Open Server Server-Library/C is used to design servers that take advantage a client/server architecture. These Open Servers access data stored in non-Sybase 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.

Topic	Page
Using Server-Library sample programs	42
Location and content	42
Tracing	43
Header files	44
Sample program summaries	44

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 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 15.5 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 in *%SYBASE%\%SYBASE_OCS%\sample\srvlib*, which 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 that your application carried out, 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 these 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 that each option provides:

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

Open Server applications require these header files:

- *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*.

Sample program summaries

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 to use the Server-Library sample programs or build applications. See Chapter 1, "Building Open Client and Open Server Applications."

Testing sample programs

Before you run any of the sample programs:

1 Verify that you can access the Adaptive Server that is configured for remote access. To do this, log in to the Adaptive Server and enter:

```
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 must be 1. If config_value is 0, enter:

```
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 about dsedit, see *Open Client and Open Server Configuration Guide for Microsoft Windows*.

3 Make sure your Open Server name exists in the Adaptive Server sysservers table. To check this, log in to Adaptive Server and enter:

```
execute sp_helpserver
```

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

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

osintro.c sample program

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

ctos.c sample program

ctos.c is a gateway Open Server application. It uses Server-Library calls and Client-Library calls. First, it accepts commands from a client and passes them to a remote Adaptive Server. Then, it retrieves the results from the remote server and passes them to the client. *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 about gateways, see the *Open Server Server-Library/C Reference Manual*.

dynlisten.c sample program

The *dynlisten.c* sample demonstrates how a running Open Server can start a new listener on a wild-carded IP port number and retrieve the final address information in a manner that can then be used to redirect logins or migrate connections.

lang.c sample program

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

See “Language Calls” in the *Open Server Server-Library/C Reference Manual*.

fullpass.c sample program

fullpass.c is an Open Server gateway application that demonstrates the Tabular Data Stream™ (TDS) passthrough mode. See "Passthrough Mode" 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.

regproc.c sample program

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

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

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

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

halang.c sample program

halang.c demonstrates how to set the HA Session ID so that an Open Client can reconnect to Open Server after a failover. Use this sample program to implement HA functionality in Open Server.

To run, execute halang *your_open_server_name* -s *failover_server &*

Note Open Server application generates the HA Session ID.

intlchar.c sample program

intlchar.c 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.

mqueue.c sample program

The *mqueue.c* sample demonstrates the use of Open Server message queues using the srv_createmsgq(), srv_putmsgq(), and srv_getmsgq() API functions. Any language command sent to the server is stored in a message queue by the language handler. A service thread logger reads messages from the queue, displays them to stdout and stores them in the log.

multthrd.c sample program

multthrd.c 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.

paramreader.c sample program

The *paramreader.c* sample program demonstrates a simple standalone Open Server application. It installs a SRV_RPC and SRV_DYNAMIC event handlers to display incoming (LOB) parameters that are received from a client application. Client examples *lobrpc.c* and *lobdynamic.c* are provided to send such parameters. These samples are mainly provided to show how to send and receive TEXT, IMAGE, or UNITEXT type parameters with RPC's or Dynamic SQL.

redirect.c sample program

redirect.c is a simple Open Server application that causes an Open Client to log into a different server than was originally specified in the interfaces file. It only works with 15.0 or later clients.

To run, execute redirect *your_open_server_name* -s *alternate_server_to_use*

Note The server directed to can be based on information within the login packet (such as the application name or server name field). The client restarts its login attempt with using the connection information provided by the redirect server.

secsrv.c sample program

secsrv.c 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, see the *Open Client and Open Server Configuration Guide for Microsoft Windows*.

sendrpc.c sample program

The *sendrpc.c* sample demonstrates sending a simple RPC command to an Adaptive Server or to an Open Server application and handling the returned results. The sample just takes simple RPC commands without parameters. For example, sp_who or ctos_shutdown.

timedsleep.c sample program

The *timedsleep.c* sample demonstrates the use of the srv_timedsleep(), srv_createmutex(), srv_lockmutex(), and srv_unlockmutex() API functions. Use the sample to play the Rock-Paper-Scissors game with two isql connections.

Note The srv_timedsleep() API function and this sample program are only available with the threaded libraries.

updtext.c sample program

The *updtext.c* sample program is used in conjunction with *uctext ctlibrary* sample. It installs a SRV_LANGUAGE event handler and responds to client commands with an informational message that contains the language command received.

Additionally, the sample also installs a bulk handler that sends a message to the client indicating the text received. This is to demonstrate the updatetext functionality. Since the server can be queried to see if the feature is supported, an rpc_handler is installed. The rpc_handler checks for the sp_mda rpc. This rpc responds to the client with metadata about the server (specifically that partial text update is supported).

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 way to retrieve, insert, or modify data stored in any Adaptive Server database.

Topic	Page
Building an Embedded SQL/C executable	53
Compiling and linking the application	54
Using Embedded SQL/C sample programs	55

Building an Embedded SQL/C executable

To build an executable program from an Embedded SQL application:

1 Precompile the application.

2 Compile the C source code generated by the precompiler.

3 Link your application to any necessary objects and libraries.

4 Load any precompiler-generated stored procedures.

Precompiling the application

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

```
cpre
    [-C compiler]
    [-D database_name]
    [-F fips_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 You can flag options using either a slash (/) or a dash (-); cpre /I and cpre -I are equivalent.

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

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

Some of the options 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).

See Appendix A, "Utility Commands 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. See the *makefile* for the actual compile-and-link syntax. The *makefile* is in *%SYBASE%\%SYBASE_OCS%\sample\esqlc*.

Link libraries

These 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, load the precompiler-generated stored procedures into Adaptive Server. To do this, 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 in *%SYBASE%\%SYBASE_OCS%\sample\esqlc*. 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.

To run the Embedded SQL/C sample programs, make sure these environment variables are set:

- SYBASE – to the path of the Sybase installation directory.
- SYBASE_OCS – to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of Open Client and Open Server version 15.5 products.

Make sure there is an entry in the *sql.ini* file 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. See the *Adaptive Server Enterprise Installation Guide* for information on installing the pubs2 database.

Before you precompile the sample programs, 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 to make the changes.

Header file

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

```
/**************************************************
 *                                                *
 *  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.

example1.cp sample program

example1.cp demonstrates how to use regular, non-scrollable cursors in an interactive query program. *example4.pc* and *example5.pc* demonstrates how to use scrollable cursors. All three of these programs:

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

example2.cp sample program

example2.cp 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.cp sample program

exampleHA.cp shows how to use Embedded SQL/C used with high availability (HA) failover capability. The program is similar to *example1.cp*, with the addition of failover processing. Error handlers to detect and handle failover.

uni_example1.cp sample program

uni_example1.cp shows how to use cursors 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 it in the character format of the client's character set.

uni_example2.cp sample program

uni_example2.cp shows how to use cursors 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 it 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 way to retrieve, insert, or modify data stored in any Adaptive Server database.

Topic	Page
Building an Embedded SQL/COBOL executable	59
Compiling and linking the application	60
Using Embedded SQL/COBOL sample programs	61

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.

Precompiling the application

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

```
cobpre
    [-Ccompiler]
    [-Ddatabase_name]
    [-Ffips_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 You can flag options using either a slash (/) or a dash (-); cobpre -I and cobpre /I are equivalent.

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

filename is the name of the Embedded SQL/COBOL source file. You must enter the name of the source file. The default extension for *filename* is *.pco*. The cobpre option generates an output file with a *.cbl* extension.

Some of the options 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).

See Appendix A, "Utility Commands 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 5.1. See the *makefile* in *%SYBASE%\%SYBASE_OCS%\sample\esqlcob* for the actual compile-and-link syntax.

Link libraries

Some or all of these 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, load the precompiler-generated stored procedures into Adaptive Server. You can use the isql -i option to accomplish this task.

For more information about isql, see Appendix A, "Utility Commands Reference."

Using Embedded SQL/COBOL sample programs

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

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.

To run the Embedded SQL/COBOL sample programs, make sure these environment variables are set:

- SYBASE – to the path of the Sybase installation directory (if it has not already been set).
- SYBASE_OCS – to the home directory of Open Client and Open Server products. For example, *OCS-15_0* is the home directory of Open Client and Open Server version 15.5 products.

You need to access an Adaptive Server on which the pubs2 sample database is installed. See the *Adaptive Server Enterprise Installation Guide* for information about 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 to make the changes.

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

Environment variables for Micro Focus COBOL

Before running the Embedded SQL/COBOL sample programs, set the environment variables in Table 6-1:

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%*

example1.pco sample program

example1.pco shows how to use regular, nonscrollable cursors in an interactive query program. The regular, nonscrollable 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

example2.pco sample program

example2.pco 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 about utility programs.

Utility	Description	Page
bcp	Bulk-copy utility, which copies a database table to or from an operating system file in a user-specified format.	65
cpre	C precompiler utility, which precompiles a C source program to produce target, listing, and isql files.	88
cobpre	COBOL precompiler utility, which precompiles a COBOL source program to produce target, listing, and isql files.	99
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.	109
isql	Interactive SQL parser, which connects to and queries an Adaptive Server or Open Server.	114

bcp

Description

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

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, 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.

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.

Note bcp raises an error and stops its operation if the number of rows to be copied in or out exceeds 2147483647.

---

datafile
The full path name of an operating system file. The path name can be 1 – 255 characters in length. To list multiple files, use a comma-separated list. The number of data files 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 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.

If the character translation files named with the -a parameter is missing, or if you mistype the names, you see:

```
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, to set the packet size to 4096 bytes for this bcp session, enter:

```
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 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 minimum value you can use 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 may be useful for locating corrupt rows of data. Use -b 1 with caution, as 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 with the -e *errorfile* to 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 cannot 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 file name information for the error is added to the error file.

Sybase recommends that you use the -e *errorfile* option with the -d *discardfileprefix* to 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, you can 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 need not interactively duplicate your previous format responses. 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 (the default is the first row). If you use multiple files, this option applies to each file.

Do not use 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 You cannot use -F 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. Standard input (stdin) is 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*.

-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 you use multiple files, 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.

If you use multiple partitions, the same number of *maxerrors* is used for every file.

-n

Performs the copy operation using native (operating system) formats. Specifying the -n parameter means bcp does 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. An indication of unrecoverable data is if you cannot re-run bcp in character format (for example, if a table was truncated or dropped, hardware damage occurred, a database table was dropped, and so on).

-N

Skips the IDENTITY column. Use this parameter when copying data in if your host data file does not include a placeholder 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. Standard output (stdout) is the default.

-P *password*

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

-Q

Provides backward compatibility with bcp 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. Either place a backslash in front of the special character or enclose it in quotes. See the examples of bcp. You need not escape special characters when running bcp in 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 that enables additional security services:

- 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 is set to CS_FALSE, and plain text (unencrypted) passwords are not used in retrying the connection.

-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 fails, 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 the -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 *%SYBASE%\%SYBASE_OCS%\ini*. 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 about 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 Adaptive Server. This does not automatically apply passwords to other encrypted columns, even if the second column is encrypted with the same key. Supply the password a second time to access the second column.

--hide-vcc

Instructs bcp not to copy virtual computed columns (VCC) either to or from a data file. 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 you use this option, Adaptive Server does not calculate or send virtual computed column data.

--initstring *"TSQL_command"*

Sends Transact-SQL commands to Adaptive Server before data is transferred.

Result sets issued by the initialization string are silently ignored, unless an error occurs. If Adaptive Server 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 Adaptive Server.

--maxconn *maximum_connections*

The maximum number of parallel connections permitted for each bulk copy operation. For example, to set the maximum number of parallel connections permitted for each operation to 2, enter:

```
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 you do not specify this parameter, 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 you see an error message.

Note You cannot use --skiprows 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. Press Return to accept 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, you need not 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 an encrypted column col1:

```
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 an external OS file named "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. bcp can read or write 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.

- bcp 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:

  ```
  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 violates 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.

- To use a previous version of bcp, 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.

- If bcp is invoked and no value is supplied for the -c, -f, or -n parameters, a bcp prompt requests the file storage type. The file storage type can be any valid Adaptive Server datatype. Storage types for the bigdatetime and bigtime Adaptive Server datatypes are specified as:

Storage type	Table datatype
A	bigdatetime
B	bigtime

- You can specify these datatypes for a bcp format file using the bigdatetime or bigtime datatypes.

Table A-1: Host file datatype storage formats

Storage format	Adaptive Server datatype
SYBBIGDATETIME	bigdatetime
SYBBIGTIME	bigtime

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, all the rows, from the beginning of the batch until the failed row are logged.
- If you use the -b option, the batch size is automatically adjusted. You see a warning message if you:

  Specify

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

Copying tables with indexes or triggers

- bcp 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 row insertions are 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 that are 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 sp_dboption DBNAME, "select into/bulkcopy", 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 need not 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. Use dump transaction to truncate the log 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 cannot 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 cannot recover your data.

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

Table A-2: Comparing fast and slow bcp

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

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

Note The performance penalty for copying data into a table that has indexes or triggers can be severe. If you are copying in a large number of rows, it may be faster to drop all the indexes and triggers; set the database option; copy the data into the table; re-create the indexes and triggers; and then dump the database. However, you must allocate extra 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 data to follow
  - 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-3 shows the defaults and possible alternate responses:

Table A-3: 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: • \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. 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 versions support 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 cannot identify which tab represents the end of the string. 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 smaller 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-4 and Table A-5 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-4: Adaptive Server char data

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

Table A-5: Other datatypes converted to char storage

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

- 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. However, if types and formats copied in are incompatible with the structure of the database table, the copy fails.
- File-storage length generally indicates the maximum amount of data to be transferred for the column, excluding terminators and prefixes.
- When copying data into a table, bcp observes any defaults defined for columns and user-defined datatypes. However, bcp ignores rules to load data at the fastest possible speed.
- Because bcp considers any data column that can contain null values to be variable length, use either a length prefix or terminator to denote the length of each data row.
- 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 in character format from a file into a database table, check the datatype entry rules in "Datatypes" in the *Adaptive Server Enterprise Reference Manual*. Character data that is copied into the database with bcp must conform to those rules. 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 files, named with the -q parameter is missing, or you mistyped the names.
- `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 these tables:
  - The table to copy
  - sysobjects
  - syscolumns
  - sysindexes

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 You can flag options using either a slash (/) or a dash (-); cpre -l and cpre /l 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 does generate strings longer than 1K.

-D *database_name*

Specifies the name of the database to parse against. Use this option to check SQL semantics at precompile time. If you specify -G, a use *database* command is 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. The precompiler can check for SQL89 or SQL92E.

-G *isql_file_name*

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 names are the input file names 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 high availability (HA) failover capability.

-I *include_path_name*

Specifies a directory complete with the path name, where Embedded SQL searches 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 in this order:

1 LC_ALL

2 LANG

If neither 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 in *%SYBASE%\locales*, 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 (default)
- syntax
- semantic

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*
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 names are the input file names 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 are assigned). If you do not use this option, the default target file name is the input file name with the extension *.cbl* appended (or replacing any input file name extension).

-P *password*
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 -G)
Specifies a tag ID (up to three 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 are assigned 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 -P[password], this option prompts you to enter a password.

Also see -K, -P, -S, and -D.

-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_155 for Open Client and Open Server version 15.5).

-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:

1. Looks for a locale name. CS-Library searches for the information in this order:
   a. LC_ALL
   b. LANG

   If neither 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 is in effect. This behavior is ANSI-compatible. See the *Adaptive Server Enterprise Reference Manual*.

-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 Embedded SQL precompiler versions 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 earlier 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.

You may find this option useful during application development but turn it 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 *%SYBASE%\%SYBASE_OCS\devlib* and *%SYBASE%\%SYBASE_OCS\devdll*, respectively.

-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. See 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; otherwise, 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 are 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

Example 1 Runs the precompiler (ANSI-compliant):

```
cpre program.pc
```

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

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

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

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

Example 4 Displays the precompiler version information only:

```
cpre -v
```

Example 5 Runs 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.
- Options work with or without a space before the argument; either is correct:
  - -Tdbg
  - -T dbg

- The precompiler can handle multiple input files. However, you may choose to not use the option -O *target_file_name*, but in doing so, you 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 are *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 are *first_input_file.lis*, *second_input_file.lis*, and so on.
- By default, cpre generates 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 runtime 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

These are the steps most commonly used in developing an Embedded SQL application. You may need to adapt this process to meet your own requirements. Perform these steps from 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:

```
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.

To choose a server on the precompile command line, use:

```
cpre -Usa -P -Sserver_name
```

As an alternative, you can leave the server name out of the connection call or statement, and *server_name* takes 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 A-6 lists the options and defaults for cpre and cobpre:

Table A-6: 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* subdirectory 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* subdirectory 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 CS_VERSION_150 for version 15.0 CS_VERSION_155 for version 15.5
-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 You can flag options using either a slash (/) or a dash (-); cobpre -I and cobpre /I 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 to check SQL semantics at precompile time. If you specify -G, a use *database* command is 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. The precompiler can check for SQL89 or SQL92E.

-G *isql_file_name*

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 names are the input file names 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 in this order:

1 LC_ALL

2 LANG

If neither 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

If you use either syntax or semantic, you must also specify the -U, -P, -S, and -D 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*
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 names are the input file names 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 are assigned). If you do not use this option, the default target file name is the input file name with the extension *.cbl* appended (or replacing any input file name extension).

-P *password*
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 -G)
Specifies a tag ID (up to three 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 are assigned 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 -P*password*, this option prompts you to enter a password.

Also see -K, -P, -S, and -D.

-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_CURRENT_ VERSION for Open Client and Open Server version 15.5).

-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:

1 Looks for a locale name. CS-Library searches in this order:

   a LC_ALL

   b LANG

   If neither 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 is in effect. This behavior is ANSI-compatible. See the *Adaptive Server Enterprise Reference Manual*.

-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 Embedded SQL precompiler versions 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 earlier 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.

You may find this option useful during application development but turn it 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 *%SYBASE%\%SYBASE_OCS%\devdll* and *%SYBASE%\%SYBASE_OCS\devdll*, respectively.

-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. See 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; otherwise, 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 you use the -y command-line option.

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

Example 1 Runs the precompiler (ANSI-compliant):

```
cobpre program.pco
```

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

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

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

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

Example 4 Displays the precompiler version information only:

```
cobpre -v
```

Example 5 Runs 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.
- Options work with or without a space before the argument; either is correct:
  - -Tdbg
  - -T dbg
- The precompiler can handle multiple input files. However, you may choose to not use the option -O *target_file_name*, but in doing so, you must accept the default target file names (see "Target file" above). If you use -G *isql_file_name*, you cannot specify an argument; the default isql file names are *first_input_file.sql*, *second_input_file.sql*, and so on. If you use -L *listing_file_name*, you cannot specify an argument; the default listing file names are *first_input_file.lis*, *second_input_file.lis*, and so on.

- By default, cobpre generates 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 runtime 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

These are the steps most commonly used in developing an Embedded SQL application. You may need to adapt this process to meet your own requirements. Perform these steps from 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:

```
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:

```
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.

To choose a server on the precompile command line, use:

```
cobpre -Usa -P -Sserver_name
```

As an alternative, you can leave the server name out of the connection call or statement, and *server_name* takes 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 A-6 for a list of options and defaults for cpre and cobpre.

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 in *%SYBASE%\%SYBASE_OCS%\bin*.

Note defncopy 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 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 or client character set is ascii_7, any 7-bit ASCII character is allowed to pass unaltered between the client and server. Other characters produce conversion errors. Character set conversion issues are discussed 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*, in *%SYBASE%\ini*.

-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.

-K *keytab_file*

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

If you do not provide the -K parameter, 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. The user must log in to the network's security system before running defncopy. 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 that enables additional security services:

- 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 fails, 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.

-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 during installation, or afterwards using 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. 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 names of database objects 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 appear 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 noninteractive way to copy 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 (more than 100 characters) that are placed before a create statement may cause defncopy to fail.

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 sp_configure. This reset restricts select permissions to the object owner and the system administrator. This restriction is required 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 Users Guide*.

- 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

isql is the interactive SQL parser to Adaptive Server. This utility is in *%SYBASE%\%SYBASE_OCS%\bin*.

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*]
[-URP *remotepassword*
[-V [*security_options*]]
[-w *column_width*]
[-W]
[-y *alternate_home_directory*]
[-z *localename*]
[-Z *security_mechanism*]
[--appname "*application_name*"]
[--conceal [':?' | '*wildcard*']]
[--help]
[--history [p]*history_length* [--history_file *history_filename*]]
[--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 or the client's character set is ascii_7, any 7-bit ASCII character is allowed to pass unaltered between client and server. Other characters produce conversion errors. Character set conversion issues are discussed 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.

-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.

--URP *remotepassword*
Enables setting the universal remote password *remotepassword* for clients accessing Adaptive Server.

-V *security_options*
Specifies network-based user authentication. With this option, the user must log in to the network's security system before running isql. 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 that enables additional security services:

- 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 fails, 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 during installation, or afterward using 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. See the description of the *libtcl.cfg* file in the *Open Client and Open Server Configuration Guide for Microsoft Windows*.

--appname "*application_name*"

Allows you to change the default application name *isql* to the isql client application name. This simplifies:

- Testing of Adaptive Server cluster routing rules for incoming client connections based on the client application name.
- Switching between alternative settings for isql in *%SYBASE%\%SYBASE_OCS%\ini\ocs.cfg*, such as between debugging and normal sessions.
- Identification of the script that started a particular isql session from within Adaptive Server.

application_name is the client application name. You can retrieve the client application name from sysprocesses.program_name after connecting to your host server.

application_name has a maximum length of 30 characters. You must enclose the entire application name in single quote or double quote characters if it contains any white spaces that do not use the backslash escape character. You can set the *application_name* to an empty string.

Note You can also set the client application name in *ocs.cfg* using the CS_APPNAME property.

--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 133.

--help

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

--history [p]*history_length* [--history_file *history_filename*]

Loads the contents of the command history log file, if it exists, when isql starts. By default, the command history feature is off. Use --history command line option to activate it.

p indicates command history persistence; in-memory command history is saved to disk when isql shuts down. If you do not use the p option, the command history log is deleted after its contents are loaded into memory.

history_length is the number of commands that isql can store in the command history log. This parameter is required if you use --history. The maximum value of *history_length* is 1024; if a larger value is specified, isql silently truncates it to 1024.

--history_file *history_filename* indicates that isql must retrieve the command history log from *history_filename*. If p is specified, isql also uses *history_filename* to store the current session's command history. *history_filename* can include an absolute or a relative path to the log file. A relative path is based on the current directory. If you do not indicate a path, the history log is saved in the current directory.

When --history_file is not specified, isql uses the default log file in *%APPDATA%\Sybase\isql\isqlCmdHistory.log*.

For information about listing, recalling, and reissuing past commands, see "Using command history" on page 132.

--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 Adaptive Server 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>
```

Example 11 Sets the application name to "isql Session 01":

```
isql -UmyAccount -SmyServer --appname "isql Session 01"
Password:
1>select program_name from sysprocesses
2>where spid=@@spid
3>go


program_name
--------------------
isql Session 01
```

Example 12 Sets the application name to the name of the script that started the isql session:

```
isql --appname $0
```

Example 13 This sample *ocs.cfg* file allows you to run isql normally or with network debug information. Because the configuration file is read and interpreted after the command line parameters are read and interpreted, setting CS_APPNAME to *isql* sets the application name back to isql:

```
;Sample ocs.cfg file
[DEFAULT]
;place holder

[isql]
;place holder

[isql_dbg_net]
CS_DEBUG = CS_DBG_NETWORK
CS_APPNAME = "isql"
```

To run isql normally:

```
isql -Uguest
```

To run isql with network debug information:

```
isql -Uguest --appname isql_dbg_net
```

Example 14 Loads and saves the command history using the default log file:

```
isql -Uguest -Ppassword -Smyase --history p1024
```

Example 15 Deletes *myaseHistory.log* after loading its contents to memory. The session's command history is not stored:

```
isql -Uguest -Ppassword -Smyase --history 1024
   --history_file myaseHistory.log
```

Example 16 Lists all the commands stored in the command history:

```
isql -Uguest -Ppassword -Smyase --history p1024
1> h
[1] select @@version
[2] select db_name()
[3] select @@servername
1>
```

Example 17 Lists the two most recent commands issued:

```
isql -Uguest -Ppassword -Smyase --history p1024
1> h -2
[2] select db_name()
[3] select @@servername
1>
```

Example 18 Recalls the command labeled 1 from the command history:

```
isql -Uguest -Ppassword -Smyase --history p1024
1> ? 1
1> select @@version
2>
```

Example 19 Recalls the latest issued command from the command history:

```
isql -Uguest -Ppassword -Smyase --history p1024
1> ? -1
1> select @@servername
2>
```

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
```

- Before you can use isql, set the SYBASE environment variable to the location of the current version of the Adaptive Server.
- The 5701 (“changed database”) server message no longer appears 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 select x = 1 100 times, enter:

```
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
  */
  ```
- 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.

Table A-7: isql session commands

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

- The command history feature is available only in command mode. Also, only commands that are issued interactively in isql are included in the command history. Examples of commands that are not included in the command history are those that are executed using the -i command line option or as part of a redirected input such as:

```
isql -Uguest -Ppassword -Smyase --history p1024
   --history_file myaseHistory.log <<EOF
exec sp_x_y_z
go
EOF
```

- Command history contains the most recent commands issued in an isql session. When *history_length* is reached, isql drops the oldest command from the history and adds the newest command issued.
- If you do not specify an alternate log file, and if the *$HOME* or *%APPDATA%* environment variable used by the default log file is not defined, an error message appears and the command history log is not saved.

In an isql session, use the h [*n*] command to display the command history. A page can display up to 24 lines of commands. If the command history contains more than 24 lines, press Enter to display the next set of commands or enter "a" to display all commands in one page. Enter "q" to return to isql.

n – indicates the number of commands to appear. If *n* is positive, the commands that appear start from the oldest command in the history. If *n* is negative, the *n* most recent commands appear.

Use the ? *n* | ?? command to recall and reissue a command from the command history.

n – when *n* is positive, isql looks for the command labeled with the number *n* and loads this to the command buffer. When *n* is negative, isql loads the *n*th most recent command issued.

?? – recalls the latest command issued and is equivalent to ? -1.

- When a command is recalled from history, the recalled command overwrites the command in the command buffer.
- You can edit a recalled command before resubmitting the command to the server.

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 Manual*.

APPENDIX B

Utility Messages

This appendix describes error, warning, and information messages for the bcp, defncopy, and isql utilities.

- bcp messages
- defncopy messages
- isql messages

bcp messages

Message 1: Memory allocation failure

Message type	Error
Symbolic constant	BERRNOMEM
Message text	`Fatal error: memory allocation failed.`
Cause	The *start* argument is invalid.
Action	Make sure the *start* argument is smaller than the length of the language or RPC command.

Message 5: Unable to open input file

Message type	Error
Symbolic constant	BERRNOINFILE
Message text	`Unable to open input file '%1!'.`
Cause	bcp could not open the data file for input.
Action	Check the specified file.

Message 6: Unable to open output file

Message type	Error
Symbolic constant	BERRNOOUTFILE
Message text	`Unable to open output file '%1!'.`
Cause	bcp could not open the data file for output.
Action	Check the specified file.

Message 7: Bad arguments

Message type	Error
Symbolic constant	BERRBADARG
Message text	`bcp: Unknown parameter '%1!'.`
Cause	An unknown parameter was submitted.
Action	Correct the unknown parameter and resubmit.

Message 8: Invalid first row

Message type	Error
Symbolic constant	BERRFIRSTROW
Message text	`When using the '%1!' flag to set the first row to copy, the row number must be greater than or equal to 1.`
Cause	The specified first row is invalid.
Action	Correct the row number.

Message 9: Invalid rows

Message type	Error
Symbolic constant	BERRFLROW
Message text	`When using the '%1!' and '%2!' flags to set the first and last rows to copy, the first row number must be smaller than the last.`

Cause	The specified first or last row is invalid.
Action	Correct the row range so that the number of the first row is lower than the number of the last row.

Message 10: Invalid last row

Message type	Error
Symbolic constant	BERRLASTROW
Message text	`When using the '%1!' flag to set the last row to copy, the row number must be greater than or equal to 1.`
Cause	Correct the row number.
Action	Make sure the *start* argument is smaller than the length of the language or RPC command.

Message 11: Invalid direction

Message type	Error
Symbolic constant	BERRBADDIR
Message text	`Copy direction must be either 'in' or 'out'.`
Cause	The direction specified is invalid.
Action	Correct the direction.

Message 12: Invalid integer

Message type	Error
Symbolic constant	BERRBADINTARG
Message text	`The '%1!' flag must be followed by an integer. '%2!' is not a legal integer.`
Cause	The argument specified is not an integer.
Action	Correct the argument.

Message 13: Duplicate flags

Message type	Warning
Symbolic constant	BERRDUPARGS
Message text	`Warning: the '%1!' flag appears more than once. The new flag's value supersedes the old.`
Cause	Duplicate arguments have been specified.
Action	Remove one of the arguments.

Message 14: Overriding arguments

Message type	Warning
Symbolic constant	BERROVERRIDE
Message text	`Warning: '%1!' overrides '%2!'`
Cause	Two arguments override each other.
Action	Remove one of the arguments if necessary.

Message 15: Invalid prefix length

Message type	Error
Symbolic constant	BERRBADPREFXLEN
Message text	`Invalid prefix length. Valid prefix-lengths are 0, 1, 2, or 4.`
Cause	The prefix length is invalid.
Action	Provide a valid prefix length.

Message 21: Retry

Message type	Information
Symbolic constant	BSTRTRY
Message text	`Try again`
Cause	A non-fatal error occurred.

Action Retry the operation.

Message 23: Starting message

Message type Information

Symbolic constant BSTRSTART

Message text

```
Starting copy...
```

Message 24: N rows copied

Message type Information

Symbolic constant BSTRROW

Message text

```
%1! rows copied.
```

Message 25: Total time

Message type Information

Symbolic constant BSTRTIME

Message text

```
Clock Time (ms.): total = %1!
```

Message 26: File save

Message type Information

Symbolic constant BSTRSAVE

Message text

```
Do you want to save this format information in a file?
[Y/n]
```

Message 27: Host file

Message type Information

Symbolic constant BSTRHOST

Message text	`Host filename [%1!]:`

Message 28: Invalid column type

Message type	Error
Symbolic constant	BERRCOLTYPE
Message text	`Invalid column type. Valid types are:`
Cause	The column type specified is invalid.
Action	Provide a valid column type.

Message 29: Invalid column type

Message type	Information
Symbolic constant	BERRDSCOL
Message text	`<cr>: same type as DataServer column.`
Cause	The column type specified is invalid.
Action	Provide a valid column type.

Message 30: Average Time

Message type	Information
Symbolic constant	BSTRAVG
Message text	`Avg = %1! (%2! rows per sec.)`
	Indicates the average processing time per row.

Message 31: Copy failure

Message type	Error
Symbolic constant	BERRCOPY
Message text	`bcp copy %1! failed`
Cause	There was an error during the copy.

Action	Retry the copy operation.

Message 32: Partial copy failure

Message type	Error
Symbolic constant	BERRPCOPY
Message text	`bcp copy %1! partially failed`
Cause	Some rows were not copied.
Action	Retry the copy operation for the specified rows

Message 33: Invalid precision

Message type	Error
Symbolic constant	BERRBADPRECISION
Message text	`Invalid precision. Precision should be between %1! and %2!`
Cause	The precision specified is invalid.
Action	Provide a valid precision.

Message 34: Invalid scale

Message type	Error
Symbolic constant	BERRBADSCALE
Message text	`Invalid scale. Scale should be between %1! and %2! and should be less than or equal to the precision.`
Cause	The scale specified is invalid.
Action	Provide a valid scale.

Message 35: Unexpected result type

Message type	Error

Symbolic constant	BERRBADTYPE
Message text	`Unexpected result type returned.`
Cause	The server returned an incorrect result type.

Message 36: Unexpected result

Message type	Error
Symbolic constant	BERRBADRESULT
Message text	`Unexpected result returned.`
Cause	The server returned an incorrect result.

Message 37: Write error

Message type	Error
Symbolic constant	BERRWRITEERR
Message text	`Error: Writing BCP file (%1!)!`
Cause	An error occurred in writing to the bcp file.
Action	Check the specified file.

Message 39: Invalid rows

Message type	Error
Symbolic constant	BERRNOTENOUGHROWS
Message text	`The first row specified is greater than the no of rows in the table.`
Cause	The number of the first row specified is greater than the number of rows in the table.
Action	Provide a valid number for the first row.

Message 40: Row transfer error

Message type	Error
Symbolic constant	BERRXFERMULT
Message text	`blk_rowxfer_mult returned unexpected return code.`
Cause	The return code from the blk_rowxfer_mult routine is unexpected.

Message 41: Invalid datatype

Message type	Error
Symbolic constant	BERRDATATYPE
Message text	`Unknown data type '%1!' encountered.`
Cause	An invalid datatype was encountered during the copy.

Message 42: Input read file error

Message type	Error
Symbolic constant	BERRIOERR
Message text	`I/O error while reading the bcp input-file.`
Cause	An error occurred in reading the bcp input file.
Action	Check the specified file.

Message 43: Error file write error

Message type	Error
Symbolic constant	BERRWEF
Message text	`I/O error while writing bcp error-file.`
Cause	An error occurred in writing to the bcp error file.
Action	Check the specified file.

Message 44: Unable to open error file

Message type	Error
Symbolic constant	BERRUOE
Message text	`Unable to open error-file.`
Cause	bcp could not open the error file.
Action	Check the specified file.

Message 45: Unexpected end-of-file

Message type	Error
Symbolic constant	BERREOF
Message text	`Unexpected EOF encountered in BCP data-file.`
Cause	There was an unexpected end-of-file character in the bcp data file.
Action	Check the specified file content.

Message 46: Negative-length prefix

Message type	Error
Symbolic constant	BERRCNL
Message text	`Negative length-prefix found in BCP data-file.`
Cause	A negative-length prefix was found in the bcp data file.
Action	Provide a valid length prefix in the bcp data file.

Message 48: Cannot read specified number of rows

Message type	Error
Symbolic constant	BERRSHORTFILE
Message text	`The BCP hostfile '%1!' contains only %2! rows. It was impossible to read the requested number of rows.`
Cause	There are fewer rows in the bcp host file than were requested to read.

Action — Specify a number of rows to read that is less than or equal to the number of rows in the bcp host file.

Message 49: Length prefix or terminator required

Message type — Error

Symbolic constant — BERRVDPT

Message text —
```
For bulk copy, all variable-length data must have either
a length-prefix or a terminator specified.
```

Cause — For bulk copy, all variable-length data must have either a length prefix or a terminator specified.

Action — Provide a length prefix or terminator.

Message 50: Text/image data truncated

Message type — Error

Symbolic constant — BERRTRUNDATA

Message text —
```
Text/image field is larger than the maximum value. Data
truncated.
```

Cause — The text or image data is larger than the maximum size. Any data beyond the maximum has been truncated.

Message 51: Max errors exceeded

Message type — Error

Symbolic constant — BERRMAXERROR

Message text —
```
The total number of errors in this BCP operation is
greater than the maximum number of errors (%1!) allowed.
BCP has stopped.
```

Cause — The bcp operation exceeded the maximum total number of errors allowed.

Message 52: Unable to open discard file

Message type	Error
Symbolic constant	BERRUOD
Message text	`Unable to open the discard-file '%1!'.`
Cause	bcp could not open the discard file.
Action	Check the specified file.

Message 53: Discard file write error

Message type	Error
Symbolic constant	BERRWDF
Message text	`I/O error while writing the bcp discard-file '%1!'.`
Cause	An error occurred in writing to the bcp discard file.
Action	Check the specified file.

Message 54: Unable to close file

Message type	Error
Symbolic constant	BERRECF
Message text	`Unable to close the file '%1!'. Data may not have been copied.`
Cause	An error occurred in closing the file.
Action	Check the specified file.

Message 55: Batch size adjusted

Message type	Warning
Symbolic constant	BERRDISCBATWAR
Message text	`Warning: Batch size adjusted to the value '%1!', for the optimization of the discard-file feature.`

Cause — The maximum memory usage has been exceeded, and the array size has been reduced.

Message 56: Max rows reached

Message type — Error

Symbolic constant — BERRMAXROWNUM

Message text —
```
The maximum row number that bcp can process is reached,
total number of '%1!' rows have been processed, bcp
operation terminated.
```

Cause — The bcp operation has processed the maximum number of rows and has been terminated.

defncopy messages

Message 1: Memory allocation failure

Message type — Error

Symbolic constant — ERRNOMEM

Message text —
```
Fatal error: memory allocation failed.
```

Cause — The Client-Library application is unable to allocate memory.

Action — Check the memory available to your application. Increase physical or virtual memory, or terminate other applications to free memory.

Message 2: Insufficient read space

Message type — Error

Symbolic constant — ERRNOREADSPACE

Message text —
```
I/O Error: Insufficient space for input data.
```

Cause — There is insufficient space in the read buffer.

Action	Check the input buffer.

Message 3: Unable to open input file

Message type	Error
Symbolic constant	ERRNOINFILE
Message text	`Unable to open input file '%1!'.`
Cause	defncopy could not open the data file for input.
Action	Check the specified file.

Message 4: Unable to open output file

Message type	Error
Symbolic constant	ERRNOOUTFILE
Message text	`Unable to open output file '%1!'.`
Cause	defncopy could not open the data file for output.
Action	Check the specified file.

Message 5: Bad argument

Message type	Error
Symbolic constant	ERRBADARG
Message text	`defncopy: Unknown parameter '%1!'.`
Cause	An unknown parameter was submitted.
Action	Provide a valid parameter.

Message 6: File not flushed

Message type	Error
Symbolic constant	ERRNOFLUSH

Message text	`Failed to flush file '%1!': '%2!'.`
Cause	The operating system failed to flush the specified file.

Message 7: Unexpected object definition

Message type	Error
Symbolic constant	ERRNOOBJDEF
Message text	`Definition of object '%1!' not found.`

Message 8: Abend

Message type	Error
Symbolic constant	ERRABORT
Message text	`defncopy aborted.`
Cause	The interrupt handler was triggered.

Message 9: Invalid direction

Message type	Error
Symbolic constant	ERRBADDIRECTION
Message text	`(direction must be either 'in' or 'out'.)`
Cause	The direction specified is invalid.
Action	Correct the direction.

Message 10: No object name

Message type	Error
Symbolic constant	ERRNOOBJNAME
Message text	`(at least one object name needed.)`
Cause	No object name was provided.
Action	Provide at least one object name.

isql messages

Message 1: Memory allocation failure

Message type	Error
Symbolic constant	LOC_ENBR_ERR_MALLOC
Message text	`Fatal error: memory allocation failed.`
Cause	The Client-Library application is unable to allocate memory.
Action	Check the memory available to your application. Increase physical or virtual memory, or terminate other applications to free memory.

Message 8: Database name length

Message type	Error
Symbolic constant	LOC_ENBR_ERR_LONGDBNAME
Message text	`Database name too long.`
Cause	The specified database name is too long.
Action	Provide a database name of valid length.

Message 9: CS-Lib message callback routine installation

Message type	Error
Symbolic constant	LOC_ENBR_ERR_CSMSGCB
Message text	`Unable to install CS-Library message callback routine.`
Cause	isql could not install an error handler.

Message 10: CT-Lib initialization

Message type	Error
Symbolic constant	LOC_ENBR_ERR_INITCTLIB
Message text	`Unable to initialize Client Library.`

Cause	A call to the ct_init function failed.
Action	Restart the application.

Message 11: CT-Lib message callback routine installation

Message type	Error
Symbolic constant	LOC_ENBR_ERR_CTMSGCB
Message text	`Unable to install Client Library client message callback routine.`
Cause	A call to the ct_callback function failed.
Action	Restart the application.

Message 12: Unsupported datatype

Message type	Error
Symbolic constant	LOC_ENBR_ERR_DATATYPE
Message text	`Unsupported datatype encountered.`
Cause	An invalid datatype was specified.
Action	Correct the invalid argument.

Message 13: Buffer overflow

Message type	Error
Symbolic constant	LOC_ENBR_ERR_BUFFOVFLW
Message text	`Buffer overflow occurred while printing row.`
Cause	There is too much data to print.

Message 15: Invalid memory block size

Message type	Error
Symbolic constant	LOC_ENBR_ERR_INVMEMBLCK

Message text	`Invalid memory block size specified.`
Cause	The Client-Library application is unable to allocate memory.
Action	Check the memory available to your application.

Message 16: Invalid memory handle

Message type	Error
Symbolic constant	LOC_ENBR_ERR_INVMEMHNDL
Message text	`Invalid memory handle specified.`
Cause	The Client-Library application is unable to allocate memory.
Action	Check the memory available to your application.

Message 17: Internal memory allocation error

Message type	Error
Symbolic constant	LOC_ENBR_ERR_INTMALLOC
Message text	`Internal isql memory allocation error.`
Cause	The Client-Library application is unable to allocate memory.
Action	Check the memory available to your application.

Message 18: Editor command too long

Message type	Error
Symbolic constant	LOC_ENBR_ERR_LONGEDCMDLN
Message text	`Command line to invoke editor too long.`
Cause	The command invoked to start the editor is too long.
Action	Reduce the command to a valid length.

Message 19: Uninitialized application context

Message type	Error
Symbolic constant	LOC_ENBR_ERR_APPCONTEXT
Message text	`An isql application context has not been initialized.`
Cause	A call to the ct_config function failed.
Action	Restart the application.

Message 20: Connection failure

Message type	Error
Symbolic constant	LOC_ENBR_ERR_CONFAILED
Message text	`A connection with a server has not been established.`
Cause	A call to the ct_connect function failed.
Action	Reattempt to connect to the server.

Message 21: Unavailable command handle

Message type	Error
Symbolic constant	LOC_ENBR_ERR_NOCMDHNDL
Message text	`No command handle is available.`
Cause	There is no command handle.
Action	Restart the application.

Message 23: File position reset failure

Message type	Error
Symbolic constant	LOC_ENBR_ERR_RSTPOSIND
Message text	`Failed to reset the file's position indicator to the beginning of the file.`
Cause	The operating system failed to reposition the indicator.

Action	Restart the application.

Message 24: Command buffer not cleared

Message type	Error
Symbolic constant	LOC_ENBR_ERR_CLRCMDBUF
Message text	`Failed to clear the command buffer.`
Cause	A call to the ct_cancel function failed.
Action	Restart the application.

Message 25: Command not initiated

Message type	Error
Symbolic constant	LOC_ENBR_ERR_INITCMD
Message text	`Unable to initiate the command.`
Cause	A call to the ct_command function failed.
Action	Restart the application.

Message 26: Command handle not cleared

Message type	Error
Symbolic constant	LOC_ENBR_ERR_CLRCMDHNDL
Message text	`Failed to clear the command in the command handle.`
Cause	A call to the ct_cancel function failed.
Action	Restart the application.

Message 28: Command argument too long

Message type	Error
Symbolic constant	LOC_ENBR_ERR_LONGCMDLN

Message text	`Command line too long.`
Cause	A command argument is too long.
Action	Provide an argument of valid length.

Message 29: Filename missing

Message type	Error
Symbolic constant	LOC_ENBR_ERR_FILENAME
Message text	`Missing file or executable name.`
Cause	A file name or executable name is missing.
Action	Provide a name for the file or executable.

Message 30: Prompt label too long

Message type	Error
Symbolic constant	LOC_ENBR_ERR_LONGPRMPTLBL
Message text	`The prompt label is too long.`
Cause	The prompt label is too long.
Action	Provide a prompt label of valid length.

Message 31: Prompt input mismatch

Message type	Error
Symbolic constant	LOC_ENBR_ERR_DIFFINPUT
Message text	`Input from 1st prompt is different from input from confirmation prompt.`
Cause	Input from the first and confirmation prompts does not match.
Action	Provide the same input for both prompts.

Message 32: Missing quote

Message type	Error
Symbolic constant	LOC_ENBR_ERR_QUOTES
Message text	`The quoted file name is missing the closing quote.`
Cause	The file name has a starting quote but no ending quote.
Action	Add the missing quote.

Message 33: Directory creation failure

Message type	Error
Symbolic constant	LOC_ENBR_ERR_PRIVDIR
Message text	`Failed to create directory '%1!': '%2!'`
Cause	isql could not create the specified directory.
Action	Check the directory containing the isql command history.

Message 34: Unexpected argument type

Message type	Error
Symbolic constant	LOC_ENBR_ERR_PRIVDIRTYPE
Message text	`Found unexpected type for '%1!': '%2!'`
Cause	isql could not determine the directory type for the command history file.
Action	Check the directory containing the isql command history.

Message 35: Unable to open history file

Message type	Error
Symbolic constant	LOC_ENBR_ERR_LOGWRITE
Message text	`Failed to open for write '%1!': '%2!'`
Cause	isql could not open the command history file for writing.
Action	Check the directory containing the isql command history.

Message 36: Temporary file deletion failure

Message type	Error
Symbolic constant	LOC_ENBR_ERR_TMPFILEDEL
Message text	`Failed to delete temporary file '%1!'`
Cause	isql could not delete the specified temporary file.

Index

A

Adaptive Server database 53

B

bcp 65, 88
  messages 135
  parameters 66, 74
bkpublic.h Bulk-Library header file 3
bkpublic.h header file 17
blktxt.c sample program 21
building a Client-Library executable 9
  example compile-and-link operations 7
  header files 2
  LIB environment variable 2
  required configuration 2
building a DB-Library executable 6
  header files 2
  link lines 9
building a Server-Library executable 1, 13
  compiling 12
  linking 12
building an Embedded SQL/C executable 53
  compiling 54
  cpre 54
  link libraries 55
  linking 54
  loading stored procedures 55, 61
  precompiling 53
  stored procedures 53
building an Embedded SQL/COBOL executable 59
building Client-Library executables
  C compilers 2
  Microsoft Windows 32-bit identifier 7
bulkcopy.c sample program 39

C

C compilers
  for Windows 2
Client-Library 27
  default values on Windows 6
  Dynamic Link Libraries (DLLs) 3
  header files 2
  sample programs 18
Client-Library sample program
  for asynchronous programming 25
  for read-only cursors 29
Client-Library sample programs 16, 19
  for asynchronous programming 20
  for bulk copy 21
  for configuration 19
  for directory services 22
  for internationalization 27
  for multithreaded programming 23
  for processing compute rows 21
  for read-only cursors 24
  for RPC commands 25
  for scrollable cursors 26, 27
  for text and image 27
  header file 17, 18
  introductory 19
  user name 18
  utility routines 19
cobpre
  defaults 98, 109
  developing an application 97, 107
  utility 88, 99
compile example
  Client-Library on Windows 7
compute.c sample program 21
configuration requirements
  sample programs 5
cpre
  options 54
  utility 88, 98, 109

CS_IFILE property 6
CS_MAX_CONNECT property 6
CS_PACKETSIZE property 6
CS-Library 15
cspublic.h header file 17
csr_disp.c sample program 24, 29
csr_disp_scrollcurs.c sample program 26
csr_disp_scrollcurs2.c sample program 27
cstypes.h header file 17
ct_callback 6
  CS_PUBLIC 6
ct_debug
  DLLs 6
ctos.c sample program 45
ctpublic.h Client-Library header file 3
ctpublic.h header file 17

D

DB-Library
  import libraries 3
DB-Library sample programs 35, 40
  for bind aggregates and compute results 35
  for browse mode and ad hoc queries 36
  for browse mode updates 36
  for bulk copy 39
  for data conversion 36
  for inserting an image 38
  for inserting data into a new table 35
  for international language routines 39
  for making an RPC call 37
  for retrieving an image 38
  for row buffering 35
  for sending a query and binding results 35
  for text and image routines 37
  for two-phase commit 39
  password 34
  user name 34
DBMAXPROS property 6
DBSETFILE property 6
debug DLLs 7
debugging 10
default values
  Client-Library on Windows 6
defncopy 113
  comments 112, 113
  messages 135
  parameters 109, 112, 113
displaying and editing rows of a table sample program 57
DLLs
  libsybblk.dll 5
  libsybcomn.dll 5
  libsybcs.dll 5
  libsybct.dll 5
  libsybdb.dll 5
  libsybintl.dll 5
  libsybsrv.dll 5
  libsybtcl.dll 5
  libsybunic.dll 5
DSLISTEN environment variable 42
dynamic link libraries (DLLs)
  Open Client and Open Server executables 3

E

Embedded SQL/C
  building an executable 53
  cpre 54
  DLL 55
  DSQUERY environment variable 98, 109
  header file 56
  linking applications 54
  loading stored procedures 55
  Open Client 53, 57
  precompiling an application 53
  pubs2 database 56
  requirements 56
  sample programs 55, 57
  Transact-SQL 53
Embedded SQL/C sample program
  for displaying and editing rows of a table 57
  for using cursors for database query 57
  for using cursors for database query with HA-Failover 58
  for using cursors for database query with unichar/univarchar support 58
  for using cursors for query of the titles table 58
Embedded SQL/COBOL 60
  building an executable 59, 61

compiling 60
cursors for database query 63
displaying and editing rows 63
executables 59
libraries 61
link libraries 60
linking 60
Open Client 63
precompiling 59
requirement 62
sample programs 61, 63
stored procedures 61
environment variables
DSLISTEN 42
INCLUDE 2
LIB 2
PATH 2
SYBASE 42
ERREXIT 3
ex_alib.c sample program 20, 25
ex_amain.c sample program 25
EX_AREAD.ME 20
ex_main.c sample program 20
EX_PASSWORD macro 18, 34
EX_USERNAME macro 18
EX_USERNAME variable 34
exampl10.c sample program 38
exampl11.c sample program 38
exampl12.c sample program 39
example.h header file 16
example1.c sample program 35
example2.c sample program 35
example3.c sample program 35
example4.c sample program 35
example5.c sample program 36
example6.c sample program 36
example7.c sample program 36
example8.c sample program 37
example9.c sample program 37
exconfig.c sample program 19
executables
building Embedded SQL/C 53
exutils.c sample program 19

F

file extensions
.c 54
.cbl 60
.pc 54
.pco 60
firstapp.c sample program 19
fullpass.c sample program 46

G

getsend.c sample program 27

H

handlers 48
SRV_ATTENTION 46
SRV_C_EXIT 49
SRV_C_RESUME 49
SRV_C_SUSPEND 49
SRV_C_TIMESLICE 49
SRV_CONNECT 46, 47, 49
SRV_LANGUAGE 46, 49
SRV_OPTION 48
SRV_START 49
header files
bkpublic.h 3, 17
Client-Library 2
cspublic.h 17
cstypes.h 17
ctpublic.h 3, 17
example.h 16
for Embedded SQL/C sample programs 56
oscompat.h 44
oserror.h 44
ospublic.h 3, 44
required for Open Server applications 44
sqlca.h 17
sybdb.h 3
syberror.h 3
sybfront.h 3
sybsqlex.h 56

I

i18n.c sample program 27
import libraries
  libsybblk.lib 4
  libsybcomn.lib 4
  libsybcs.lib 4
  libsybct.lib 4
  libsybdb.lib 4
  libsybsrv.lib 4
INCLUDE environment variable 2
intlchar.c sample program 47
isql 133
  character set input 117
  comments 123, 131
  examples 75, 121
  filters 117
  messages 135
  parameters 120, 132
  stored procedures 55

L

lang.c sample program 46
LIB environment variable 2
libcobct file 61
libcomn file 61
libcs file 61
libct file 61
libintl file 61
libraries
  Embedded SQL/C 55
  Embedded SQL/COBOL 61
libsybblk.dll file 5
libsybblk.lib file 4
libsybcomn.dll file 5
libsybcomn.lib file 4
libsybcs.dll file 5
libsybcs.lib file 4
libsybct.dll file 5
libsybct.lib file 4
libsybdb file 5
libsybdb.lib file 4
libsybintl.dll file 5
libsybsrv.dll file 5
libsybsrv.lib file 4
libsybtcl.dll file 5
libsybunic.dll file 5
libtcl file 61

M

messages
  bcp 135
  defncopy 135
  isql 135
modes
  scheduling 10
multithread programming
  support for Windows 7
multthrd.c sample program 23, 48

O

Open Server sample programs 44, 49
  for international languages and character sets 47
  for language event handler 46
  for multithreaded features 48
  for Open Server gateway 45
  for registered procedures 47
  for security services 49
  for TDS passthrough mode 46
  introductory 45
  location 42
oscompat.h header file 44
oserror.h header file 44
osintro.c sample program 45
ospublic.h header file 44
ospublic.h Server-Library header file 3

P

PATH environment variable 2, 4
precompilers
  cobpre 59
  cpre 54
  determining servers 97, 108
  for Embedded SQL/C 53
  for Embedded SQL/COBOL 59

preemptive mode
  scheduling 10
  srv_sleep 11
  Windows programming 10, 11
programming issues for Client-Library 7
  ct_callback 6
programming issues for Client-Library on Windows 6
programming issues for Server-Library 10
  scheduling modes 10
  srv_callback 10
properties
  CS_IFILE 6
  CS_MAX_CONNECT 6
  CS_PACKETSIZE 6
  DBSETFILE 6
  DBSETMAXPROS 6
pubs2 database 56

R

regproc.c sample program 47
requirements
  configuration 5
  Embedded SQL/C sample programs 56
rpc.c sample program 25

S

sample programs
  Client-Library 19
  DB-Library 35, 40
  Open Server 44, 49
sample programs for Embedded SQL/C 55, 57
  displaying and editing rows of a table 57
  header file 56
  requirements 56
  using cursors for database query 57
scheduling mode 10
  srv_sleep 11
  srv_wakeup 11
secsrv.c sample program 49
Server-Library
  compile example 12
  link example 12
  programming issues 10
servers
  precompilers 97, 108
sql.ini file 6
sqlca.h header file 17
srv_callback 10
srv_sleep 10
srv_wakeup 11
STDEXIT 3
stored procedures 53, 54, 59, 61
  for Embedded SQL/C 55
  isql 55
  loading 55, 61, 97, 108
SYBASE environment variable 42, 56, 62
sybdb.h DB-Library header file 3
syberror.h DB-Library header file 3
sybfront.h DB-Library header file 3
sybsqlex.h header file 56

T

thrdfunc.c sample program 23
tracing 43
  options 43
Transact-SQL 53, 59
twophase.c sample program 39

U

usedir.c sample program 22
using cursors for database query sample program 57
using cursors for database query with HA-Failover sample program 58
using cursors for database query with unichar/univarchar support sample program 58
using cursors for query of the titles table sample program 58
utilities
  bcp 65, 88
  cobpre 99
  cpre 88, 98, 109
  defncopy 113
  isql 133

W

Windows
- building a Client-Library executable 9
- building a DB-Library executable 6
- building a Server-Library executable 1, 13
- C compilers 2
- multithreaded programming support 7

Windows properties
- Client-Library 6
- CS_IFILE 6
- CS_MAX_CONNECT 6
- CS_PACKETSIZE 6
- DBMAXPROS 6
- DBSETFILE 6