SYBASE®

Programmer's Reference for PL/1

Mainframe Connect™ Client Option

15.0

IBM CICS, IMS, and MVS

DOCUMENT ID: DC36460-01-1500-01

LAST REVISED: July 2007

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

Contents

About This Book

This book contains reference information for the PL/1 version of Open ClientConnect™ Client-Library.

Note The Open ClientConnect Client-Library is a subset of the generic Sybase® Client-Library.

Audience

This guide is a reference manual for mainframe programmers who write client applications in the PL/1 programming language using Open ClientConnect Client-Library.

This manual assumes that the programmer is familiar with the PL/1 programming language and knows how to write programs under either CICS or IMS TM. This book does not contain instructions for writing PL/1 programs. Rather, it describes the functions that can be called within your PL/1 programs to enable them to communicate with remote servers.

How to use this book

This book includes these chapters:

- Chapter 1, "Open ClientConnect Processing," provides an overview of Open ClientConnect including discussion of different kinds of client requests and explanations of how Open ClientConnect programs process them.

  Note Everyone who writes programs using Open ClientConnect should read this chapter.

- Chapter 2, "Topics," describes Gateway-Library concepts, and information on how to accomplish specific programming tasks. This chapter discusses tasks, resources, and other topics that the application programmer needs to understand to write Gateway-Library applications. It includes a detailed discussion of the Gateway-Library cursor, dynamic SQL and Japanese language support and a list of supported datatypes and models for structures used to store data.

- Chapter 3, "Functions," contains reference pages for each Gateway-Library function. Each function description contains sections on functionality, syntax, explanatory comments and related functions, as well as an example.
- Appendix A, "Sample Language Application," contains a sample PL/1 application program that processes client language requests under CICS.
- Appendix B, "Sample RPC Application," contains a sample PL/1 application program that processes client RPC requests under CICS.
- Appendix C, "Sybase Product Documentation by Audience," contains a list and description of relevant Sybase product documentation by audience.

Product name changes

The following table describes new names for products in this release of the Mainframe Connect™ Integrated Product Set.

Old product name	New product name
Open ClientConnect for CICS	Mainframe Connect Client Option for CICS
Open ClientConnect for IMS and MVS	Mainframe Connect Client Option for IMS and MVS
Open ServerConnect™ for CICS	Mainframe Connect Server Option for CICS
Open ServerConnect for IMS and MVS	Mainframe Connect Server Option for IMS and MVS
Mainframe Connect for DB2 UDB	Mainframe Connect DB2 UDB Option for CICS
DirectConnect™ for OS/390	DirectConnect for z/OS

The old product names are used throughout this book, except for on the title page.

Note This book also uses the terms MVS and OS/390 where the newer term z/OS would otherwise be used.

Related documents

The documentation set consists of:

- The *Release Bulletin* for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Product Manuals Web site.

- Mainframe Connect Server Option for CICS *Installation and Administration Guide* – describes configuring the Mainframe Connect network, installing Open ServerConnect, setting up security, and troubleshooting for an MVS-CICS environment.
- Mainframe Connect Server Option for IMS and MVS *Installation and Administration Guide* – describes configuring the Mainframe Connect network, setting up APPC communications, installing Open ServerConnect, setting up security, and troubleshooting for an IMS or MVS environment.
- Mainframe Connect Client Option for CICS *Installation and Administration Guide* – describes installing and configuring Open ClientConnect, routing requests to a server, and using Sybase isql. This manual also contains instructions for using the connection router and the mainframe-based isql utility.
- Mainframe Connect Client Option for IMS and MVS *Installation and Administration Guide* – describes installing Open ClientConnect, routing requests to a server, and using Sybase isql. This manual also contains instructions for using mainframe-based isql utility.
- Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* – describes configuring the mainframe, installing MainframeConnect, setting up security, and troubleshooting for a CICS environment.
- Mainframe Connect DirectConnect for z/OS Option *Installation Guide* – describes installing a DirectConnect server and service libraries.
- Enterprise Connect™ Data Access and Mainframe Connect *Server Administration Guide* for DirectConnect – describes administration of the DirectConnect server. Information about administering specific service libraries and services is provided in other DirectConnect publications.
- Mainframe Connect Client Option *Programmer's Reference for PL/1* – describes writing Open ClientConnect programs that call PL/I Client-Library functions. This guide contains reference pages for Client-Library routines and descriptions of the underlying concepts for PL/I programmers.
- Mainframe Connect Server Option *Programmer's Reference for PL/1* – provides reference material for writing Open ServerConnect programs that call PL/I Gateway-Library functions. This guide contains reference pages for Gateway-Library routines and descriptions of the underlying concepts for PL/I programmers.

- Mainframe Connect Client Option *Programmer's Reference for COBOL* – describes writing Open ClientConnect programs that call COBOL Client-Library functions. This guide contains reference pages for Client-Library routines and descriptions of the underlying concepts for COBOL programmers.
- Mainframe Connect Server Option *Programmer's Reference for COBOL* – provides reference material for writing Open ServerConnect programs that call COBOL Gateway-Library functions. This guide contains reference pages for Gateway-Library routines and descriptions of the underlying concepts for COBOL programmers.
- Mainframe Connect Client Option *Programmer's Reference for C* – describes writing Open ClientConnect programs that call C Client-Library functions. This guide contains reference pages for Client-Library routines and descriptions of the underlying concepts for C programmers.
- Mainframe Connect Server Option *Programmer's Reference for RSPs* – provides information for anyone who designs, codes, and tests remote stored procedures (RSPs).
- Mainframe Connect Client Option *Programmer's Reference for CSAs* – provides information for anyone who designs, codes, and tests client services applications (CSAs).
- Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services* – describes configuring, controlling, and monitoring DirectConnect Transaction Router Service Library, as well as setting up security.
- Mainframe Connect DirectConnect for z/OS Option *User's Guide for DB2 Access Services* (for use with MainframeConnect for DB2 UDB) – describes configuring, controlling, and monitoring DirectConnect for OS/390 Access Service, as well as setting up security.
- Mainframe Connect Client Option and Server Option *Installation and Administration Guide* – Provides details on messages that Mainframe Connect components return.

Other sources of information

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

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

- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

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

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

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

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

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by 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

This section describes the syntax and style conventions used in this book.

Note Throughout this book, all references to Adaptive Server® Enterprise also apply to its predecessor, SQL Server. Also, Adaptive Server Enterprise (ASE) and Adaptive Server (AS) are used interchangeably.

The Client Option uses eight-character function names, while other versions of Client-Library use longer names. This book uses the long version of Client-Library names with one exception: the eight-character version is used in syntax statements. For example, CTBCMDPROPS has eleven letters. In the syntax statement, it is written CTBCMDPR, using eight characters. You can use either version in your code.

Table 1 explains syntax conventions used in this book.

Table 1: Syntax conventions

Symbol	Explanation
( )	Parentheses indicate that parentheses are included as part of the command.
{ }	Braces indicate that you must choose at least one of the enclosed options. Do not type the braces when you type the option.
[ ]	Brackets indicate that you can choose one or more of the enclosed options, or none. Do not type the brackets when you type the options.
|	The vertical bar indicates that you can select only one of the options shown. Do not type the bar in your command.
,	The comma indicates that you can choose one or more of the options shown. Separate each choice by using a comma as part of the command.

Table 2 explains style conventions used in this book.

Table 2: Style conventions

This type of information	Looks like this
Gateway-Library function names	TDINIT, TDRESULT
Client-Library function names	CTBINIT, CTBRESULTS
Other executables (DB-Library routines, SQL commands) in text	the dbrpcparam routine, a select statement
Directory names, path names, and file names	*/usr/bin directory, interfaces* file
Variables	*n* bytes
Adaptive Server datatypes	datetime, float
Sample code	01 BUFFER PIC S9(9) COMP SYNC. 01 BUFFER PIC X(n).
User input	01 BUFFER PIC X(n)
Client-Library and Gateway-Library function argument names	*BUFFER, RETCODE*
Client-Library function arguments that are input (I) or output (O)	*COMMAND* – (I) *RETCODE* – (O)
Names of objects stored on the mainframe	SYCTSAA5
Symbolic values used with function arguments, properties, and structure fields	CS-UNUSED, FMT-NAME, CS-SV-FATAL

This type of information	Looks like this
Client-Library property names	CS-PASSWORD, CS-USERNAME
Client-Library and Gateway-Library datatypes	CS-CHAR, TDSCHAR

All other names and terms appear in this typeface.

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.

The HTML 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/products/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

Open ClientConnect Processing

This chapter includes the following topics:

What is Open ClientConnect?

Open ClientConnect is a programming environment that provides Open Client Client-Library routines for use in building mainframe client applications.

Open ClientConnect runs on an IBM System/390 or plug-compatible mainframe computer. It uses the LU 6.2 or TCP/IP communications protocols and is available for CICS, IMS TM, and native MVS host transaction processors.

Note Some information in this guide is specific to version 4.0 of Open ClientConnect. This information will apply to Open ClientConnect for CICS only. All other information will apply to Open ClientConnect for CICS and Open ClientConnect for IMS and MVS.

Open ClientConnect applications can communicate with two kinds of servers:

- Adaptive Server Enterprise and Open Server on PCs and several mid-range UNIX platforms
- Open ServerConnect applications running in a separate region on the mainframe

Open ClientConnect applications can send requests to Adaptive Server Enterprise, Open ServerConnect™ applications, and Mainframe Connect for DB2 UDB (or OmniSQL Access Module™ for DB2).

Adaptive Server Enterprise

Open ClientConnect applications can send requests to Adaptive Server Enterprise indirectly through either of the following:

- The three-tier (gateway-enabled) environment using Mainframe ClientConnect (MCC).
- The two-tier (gateway-less) environment using TCP. See Two-tier (gateway-less) environments for more information on two-tier environments.

Open ServerConnect

Open ClientConnect applications can send requests directly to Open ServerConnect running in a different CICS region. If using TCP, Open ClientConnectmay send requests to Open ServerConnect running in the same CICS region.

Note Due to an IBM SNA restriction, connections from Open ClientConnect to Open ServerConnect require that they reside in different regions when connecting through LU 6.2. For TCP/IP, they can reside in the same region.

Understanding three-tier and two-tier environments

Open ClientConnect supports both three-tier (gateway-enabled) and two-tier (gateway-less) environments. When installing and using Open ClientConnect, follow the instructions in this book that are specific to your environment.

Three-tier (gateway-enabled) environments

If you use SNA as your protocol, you must use a three-tier environment for routing.

Note The DirectConnect product no longer comes with an MCC for TCP. A three-tier (gateway-enabled) environment using TCP as your protocol is no longer an option.

Two-tier (gateway-less)

If you have standardized to TCP for connectivity from CICS to Adaptive Server Enterprises, you must use the two-tier environment for routing. The two-tier environment allows Open ClientConnect to directly login to an Adaptive Server Enterprise, which eliminates the need for an MCC gateway.

An Open ClientConnect network configuration using two-tier (gateway-less) processing consists of the following:

- A host-based client, which is an Open ClientConnect program running under CICS. The client program selects a server and sends requests to that server.
- A server, which can be any server that Open ClientConnect applications can access, including servers on the LAN, Sybase Open Servers and Adaptive Server Enterprises, as well as Open ServerConnect running in a separate CICS region. If you are using TCP as your protocol, Open ClientConnect may access servers running in the same CICS region.

Open ClientConnect communications

This section describes Open ClientConnect communications in three-tier and two-tier environments. It also explains the communication flow for both environments.

Note Although it is not shown in any of the figures in this section, Open ClientConnect for CICS also works with Open ServerConnect for IMS and MVS.

Three-tier (gateway-enabled) environments

The following figures show a basic Open ClientConnect configuration for CICS in three-tier (gateway-enabled) SNA and TCP/IP environments:

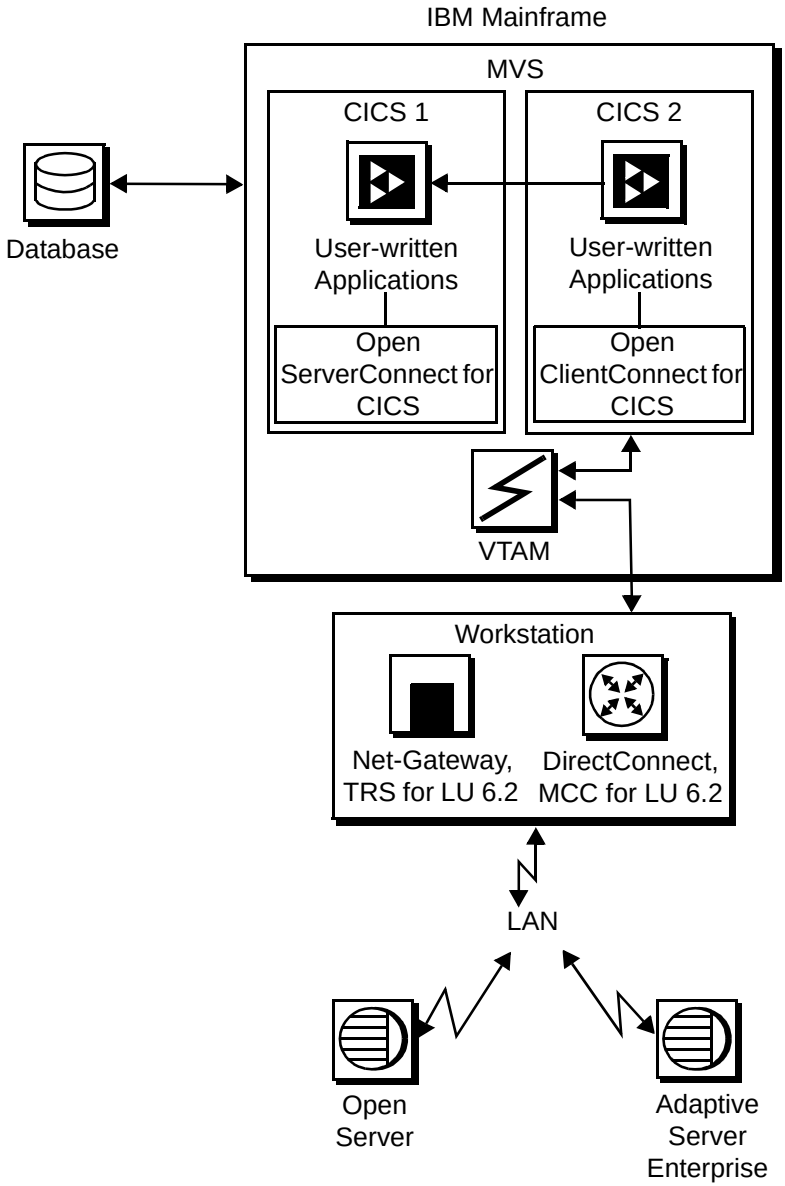
- Figure 1-1 on page 5
- Figure 1-2 on page 6

Note For three-tier, gateway-enabled environments, DirectConnect 11.1 (or Net-Gateway version 3.0.1 or higher) is a required companion product for full-feature compatibility with Open ServerConnect version 4.0 and Open ClientConnect version 3.2.

Three-tier SNA environments

Figure 1-1 shows Open ClientConnect communication in a three-tier (gateway-enabled) SNA environment.

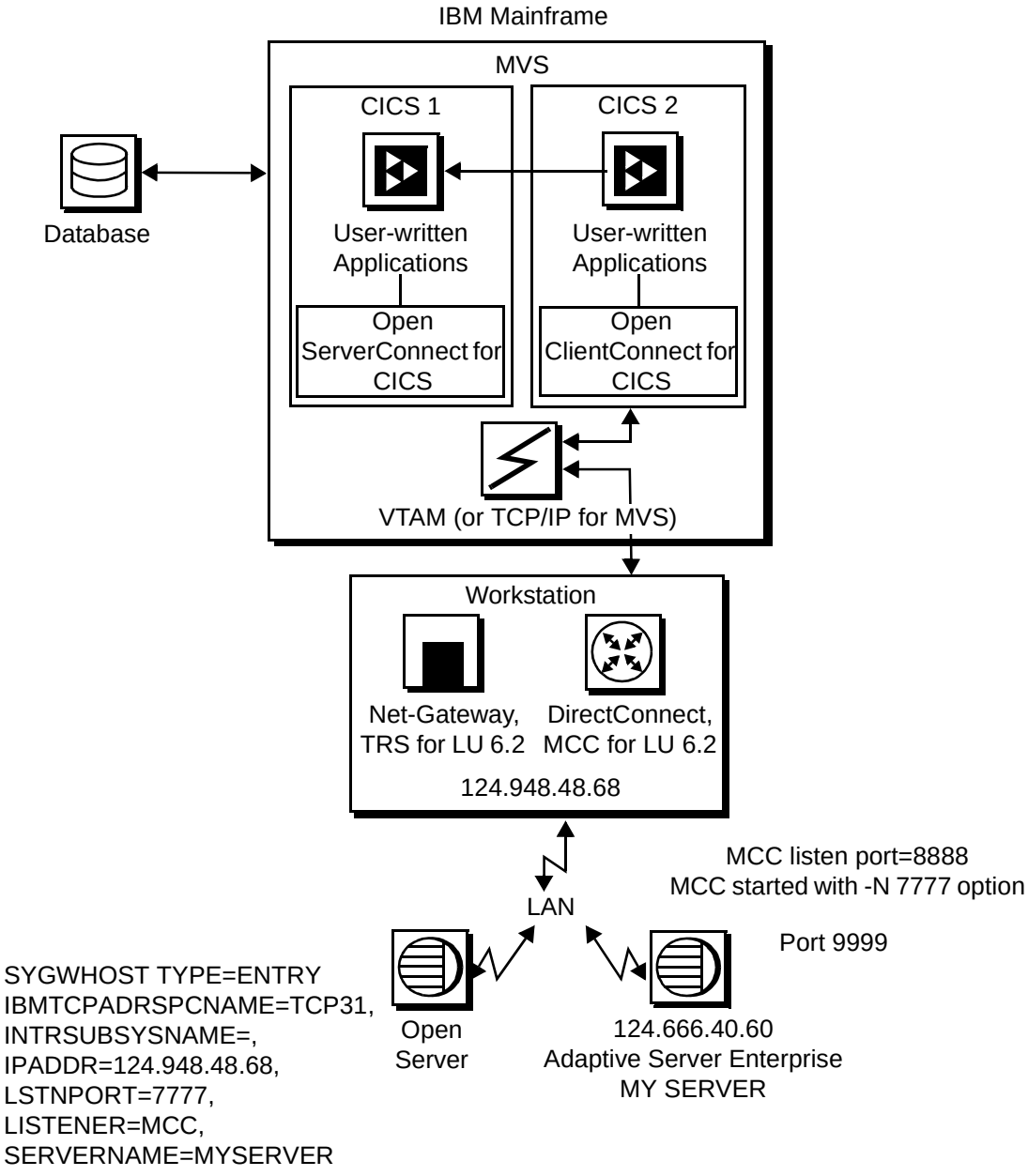
Figure 1-1: Open ClientConnect in a three-tier SNA environment



Three-tier TCP environments

Figure 1-2 shows Open ClientConnect communication in a three-tier (gateway-enabled) TCP environment.

Figure 1-2: Open ClientConnect in a three-tier TCP environmenT

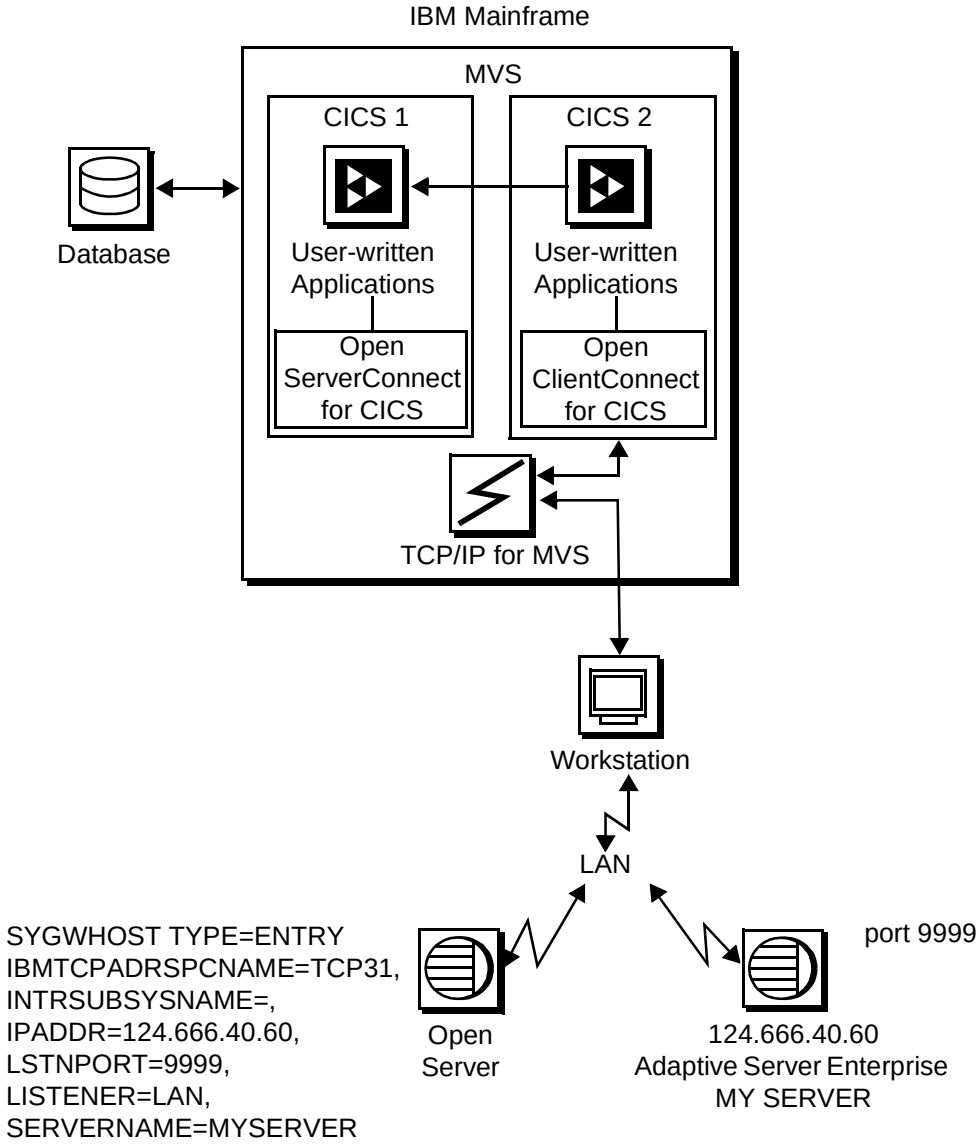


Two-tier (gateway-less) environments

Two-tier TCP environments

Figure 1-3 shows Open ClientConnect communication in a two-tier (gateway-less) TCP environment.

Figure 1-3: Open ClientConnect in a two-tier TCP environment



Communication flow

This section describes what happens at the mainframe, at the DirectConnect installation, and at the server in Open ClientConnect processing.

At the mainframe

An Open ClientConnect application calls a pre-written procedure, such as a stored procedure or an Open ServerConnect application. All calls from Open ClientConnect to remote nodes are processed using the LU 6.2 (three-tier only) or TCP/IP (two-tier only) communications protocol. For requests to an Open Server, the client can access any data available to the Open Server application. If the request is to Open ServerConnect, the client can access any data storage system accessible through CICS.

The called procedure or transaction executes and returns results to the calling Open ClientConnect application, which can use the results for local processing. If the client has permission, the client transaction can update data at remote sites by inserting, modifying, and deleting entries in database tables or other data storage systems. For additional information on any of the following topics, refer to the Client Option for CICS Installation and Administration Guide.

isql utility

Open ClientConnect includes isql, a utility that allows users to send SQL language commands interactively. Users specify the server, whether or not to enable tracing, and type SQL commands in a 3270 panel.

Connection Router Table (SNA Only)

For SNA environments, Open ClientConnect includes a Connection Router Table that allows you to define servers and connections, and to set traffic priorities.

Server-Host Mapping Table (TCP/IP only)

For TCP/IP, Open ClientConnect includes a Server-Host Mapping Table that allows you to define servers for both three-tier and two-tier environments.

Side Information (SNA only)

Open ClientConnect for CICS uses the APPC Side Information File to define servers.

At the server

Typically, a server accepts requests from a client and returns results. In an Open ClientConnect environment, the server can be an Adaptive Server Enterprise, an Open Server, or Open ServerConnect on the mainframe.

From the server standpoint, a request from an IBM host is no different than a request from a Sybase client. Open ClientConnect participates in ASCII-EBCDIC translations and datatype conversions.

Open ClientConnect security

Security for Open ClientConnect processing can be configured to require permission to:

- Log into the target server or desired CICS region
- Use specific commands, stored procedures or transactions, and data objects at the target server

For more security-related information regarding:

- Adaptive Server Enterprise, refer to the chapter called “Security Administration,” in the Adaptive Server Enterprise *System Administration Guide*
- DirectConnect, refer to the Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*
- Mainframes, refer to documentation provided with CICS and MVS, or the appropriate mainframe security system

How to choose a network driver

Open ClientConnect supports concurrent use of multiple network drivers, providing additional flexibility and ease of installation for sites configured to run mixtures of SNA and TCP/IP.

Note Dynamic network driver support is a new feature in Open ClientConnect version 4.0.

The network drivers can be invoked from the same Open Client/Open Server common code base. The appropriate network driver is loaded dynamically at the time the program executes.

You must use the SYGWDRIV macro to define the network drivers to be used with Open ClientConnect and Open ServerConnect. For each operating environment (CICS and MVS), the default SYGWXCPH member provided contains the SYGWDRIV macro definitions for all the supported network drivers pertinent to the technology. The person installing Open ClientConnect should edit the appropriate SYCTCUST member to comment-out the drivers that your site does not intend to use.

This section provides an overview of network communication, describes general criteria for choosing a driver, and explains how to choose between a CPI-C/LU 6.2 and a TCP/IP driver.

Overview of network communication definitions

You need to choose which dynamic network drivers to use at your site. Your choice depends on the protocols installed at your site and the types of processing you want to achieve.

Use this overview to understand issues involved in selecting your drivers.

- System Application Architecture (SAA)
- Common Programming Interface (CPI)
- APPC/MVS
- Systems Network Architecture (SNA)
- LU 6.2
- Advanced Program-to-Program Communications (APPC)
- Common threads between APPC MVS, CICS, and IMS TM

System Application Architecture (SAA)

SAA is an architecture composed of a set of selected software interfaces, conventions, and protocols designed to provide a framework for developing distributed applications. The key benefits of SAA are portability, consistency, and connectivity. The components of SAA are specifications for the key application interfaces points:

- Common user access
- Common communication support
- Common Programming Interface (CPI)

Common Programming Interface (CPI)

The SAA Common Programming Interface specifies the languages and services used to develop applications across SAA environments. The elements of the CPI specification are divided into two parts:

- Processing logic
  - High level language – COBOL, C, Fortran, RPG
  - Procedure language – REXX
  - Application generator – Cross Systems Product/Application Development (CSP/AD)

- Services
  - Communication Interface or CPI-C – API for writing APPC applications.
  - Database interface – Structured Query Language (SQL)
  - Dialog interface – Interactive System Productivity Facility (ISPF)

APPC/MVS

APPC/MVS is an SNA application that extends APPC support to the MVS operating system. The primary role of APPC/MVS is to provide full LU 6.2 capability to MVS applications to allow communication with other applications in a distributed SNA network.

APPC/MVS provides programming support by providing an API based on the CPI-C interface. This interface is implemented in a lower-level API that is MVS-specific. The CPI-C calls all begin with CM; for example, CMALLC (Allocate). The MVS calls all begin with ATB; for example, Send_data (ATBSEND). The CPI-C calls are portable to non-MVS platforms while the ATB calls are not portable to non-MVS platforms.

Systems Network Architecture (SNA)

SNA is an IBM Network Architecture composed of a set of software interfaces, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

LU 6.2

LU 6.2 refers to the SNA Logical Unit Type 6.2, which supports general communication between programs in a distributed environment. LU 6.2 is characterized by peer-to-peer communications support, comprehensive end-to-end error processing, optimized data transmission flow and a generic API.

The LU 6.2 system is layered functionally. It can be represented by a set of finite-state machines. Each of these machines has a finite number of states and a set of rules that govern the transition from one state to another. These finite state machines govern the behavior of LU 6.2 devices by guaranteeing that a given input always produces the same output.

Advanced Program-to-Program Communications (APPC)

APPC is peer-level data communication support based on the SNA LU 6.2 protocols.

Common threads between APPC MVS, CICS, and IMS TM

All inbound transactions require a scheduler. Under MVS, the ASCH address space performs this function by scheduling inbound transactions in initiators under its control. The relationship between ASCH and its initiators is very similar to that of JES (Job Entry System), which schedules jobs in initiators under its control.

The Control region is the scheduler running under CICS. The Message Region running under the Control region corresponds to the initiators used by ASCH.

CICS differs from MVS and IMS TM because it does not schedule transactions in a separate address space. It schedules them as a task within its own address space.

Outbound transactions use a file called the Side Information File to map a name to an SNA logical unit. MVS and IMS TM both use this file.

General criteria for choosing a driver

The choice of a network driver depends on several factors:

- Network type –SNA or TCP/IP
- Network environment – two-tier (TCP/IP only) or three-tier (SNA only)
- Operating environment – CICS or MVS

This section explains why you might want to choose a particular driver in each environment.

Network type and environment

Because non-MVS platforms do not support LU 6.2 in their operating systems, if your network type is SNA, then you *must* use a three-tier network environment with a gateway. In a three-tier (gateway-enabled) environment, you must use either an LU 6.2 or CPI-C driver.

If your network type is TCP/IP, your network environment must be two-tier (gateway-less). The choice between IBM TCP/IP and Interlink depends on which product is installed in the network environment, although Open ClientConnect supports both concurrently.

Operating environment

This section explains the drivers used in CICS and MVS environments.

CICS environment

The following drivers are supported in the CICS environment:

- LU 6.2
- CPI-C
- IBM TCP/IP
- Interlink TCP/IP

The LU 6.2 driver only supports incoming transactions sent to the CICS message queue. It does not support Open Client outbound requests from the mainframe. With the LU 6.2 driver, CICS builds an entire result set in the message queue and sends that entire result set to the MSG.

The CPI-C driver supports both Open Client and Open Server requests. It does not use the CICS message queue to send or receive requests. Therefore, result sets sent by Open Server using the CPI-C driver can be interrupted. However, with the LU 6.2 driver, if a client does a CTRL-C to cancel the result set, the gateway must read the entire result set and throw it away.

MVS environment

The following drivers are supported in the MVS environment:

- CPI-C
- IBM TCP/IP
- Interlink TCP/IP

Choosing between a CPI-C/LU 6.2 driver and a TCP/IP driver

When choosing between a CPI-C/LU 6.2 driver and a TCP/IP driver, consider the following factors:

- Network type – SNA or TCP/IP
- Reliability
- Performance
- Network operating environment - Two or three-tier

Network type

If your current network is SNA-only or TCP/IP-only, choose the driver that supports your network protocol.

Reliability

SNA networks have been running on IBM mainframes much longer than TCP/IP based-networks, and the SNA operational procedures are well established. However, if your LAN-side staff is not very familiar with SNA on a particular vendor platform, the SNA setup can be difficult.

TCP/IP is simpler to set up and maintain from the LAN, although it can be a challenge to get it running under MVS for the first time. In addition, TCP/IP on mainframes is a relatively new technology compared to SNA and as such, SNA is probably more robust and reliable.

Performance

TCP/IP performance appears to equal and in some cases exceed SNA-based performance. When going from the LAN to a mainframe, SNA requires a gateway, while TCP/IP does not.

Network operating environment

Small, two-tier (gateway-less) Client Server networks are easier to set up and maintain than three-tier (gateway-enabled) networks, because three-tier networks have a gateway between the mainframe and the LAN. However, three-tier networks scale better, provide a single point of entry for security, and provide tracing facilities that can be easily enabled.

Compatibility

For full functionality with the current version, use these mainframe access components listed in Table 1-1, as available at your site.

Table 1-1: Open ClientConnect version compatibility

Component	Version level
Open ServerConnect	3.1 or higher
Open ClientConnect	3.2 or higher
MainframeConnect for DB2 UDB	11.1 or higher
or OmniSQL Access Module for DB2	10.1 with latest Emergency Bug Fix (EBF)
DirectConnect Transaction Router Service	11.1
or Net–Gateway	3.0.1 for full functionality
Japanese Conversion Module	3.1

Open ClientConnect Client-Library functions

Open ClientConnect includes a programming interface of functions that are used in writing mainframe client applications. Some of these functions prepare and send requests to a server; others retrieve and process the results. Additional functions set application properties, handle error conditions, and provide a variety of information about an application's interaction with a server.

Open ClientConnect's Client-Library functions are similar in name and function to Open Client Client-Library's routines. However, not all Sybase Open Client routines are supported. For example, Open ClientConnect does not currently support cursors or compute rows.

Most mainframe Client-Library function names begin with "ct_"; for example, the function that exits from Client-Library is named ct_exit. This corresponds to the Client-Library/C routine ct_exit.

Open ClientConnect includes utility functions which allocate, drop, and assign properties to context handles, and one function used in datatype conversion. These functions begin with the prefix: "cs_" instead of "ct_." This naming convention derives from related Sybase products, but it is irrelevant for Open ClientConnect. Use cs_xxxxx functions just as if they were ct_xxxxx functions.

Structures, types, and values used by Client-Library functions are defined in header files.

Client-Library functions are described in detail in Chapter 3, "Functions."

Using Client-Library functions

An application programmer writes a client program, adding calls to Client-Library functions to set up control structures, connect to servers, send commands, process results, and clean up. A Client-Library program is compiled, linked, and run in the same way as any other PL/1 program under CICS or IMS TM.

Note An application program can act as both client and server. Such a program, called a *mixed-mode* program, contains both Client-Library calls to send requests and Gateway-Library calls to accept and process requests.

Basic control structures

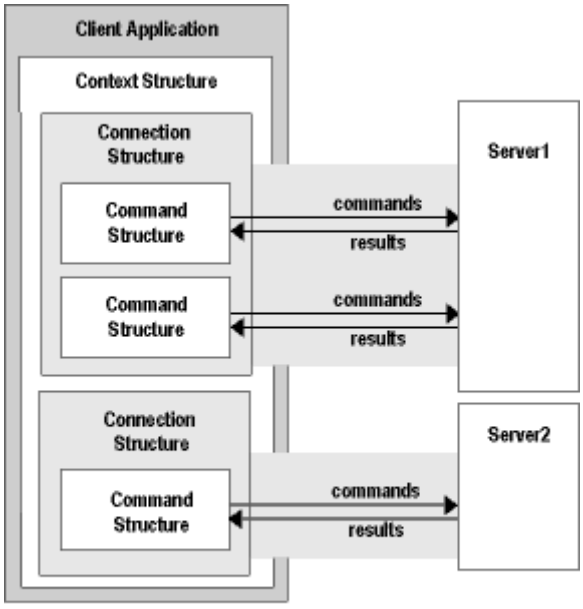
In order to send commands to a server, an application must allocate three types of structures:

- A context structure, which defines a particular application context, or operating environment.
- A connection structure, which defines a particular client/server connection.
- A command structure, which defines a "command space" in which commands are sent to a server.

An application allocates these structures by calling the functions cs_ctx_alloc, ct_con_alloc, and ct_cmd_alloc.

The relationship between these control structures is illustrated in Figure 1-4.

Figure 1-4: Client-Library control structures



Through these structures, an application sets up its environment, connects to servers, sends commands, and processes results.

Note An Open ClientConnect application is restricted to one context per application. This differs from applications written in other versions of Client-Library, which support multiple context structures.

Steps in a simple program

A simple program involves the following steps:

1 Set up the programming environment:

cs_ctx_alloc – allocate a context structure.
ct_init – initialize the programming interface.

2 Establish a connection with a server or CICS or IMS region:

ct_con_alloc – allocate a connection structure.
ct_con_props – set or retrieve connection structure properties.
ct_connect – connect to a server.

3 Send a command to the server or to a CICS or IMS region.

(For three-tier processing, send a command to Mainframe ClientConnect which forwards the request to the target server, and routes the results back to the client program.)

ct_cmd_alloc – allocate a command structure.
ct_command – initiate a language request or remote procedure call.
ct_send – send a request to the server.

4 Process the results of the command:

ct_results – set up result data to be processed.
ct_res_info – return result set information.
ct_bind – bind a returned column or parameter to a program variable.
ct_fetch – fetch result data.

5 Finish up:

ct_cmd_drop – deallocate a command structure.
ct_close – close a server connection.
ct_con_drop – deallocate a connection structure.
ct_exit – exit the programming interface.
cs_ctx_drop – deallocate a context structure.

A simple language program

The following walk through demonstrates the basic framework of an Open ClientConnect application. The program follows the steps outlined in the previous section, sending a language request to an Adaptive Server and processing the results. In this case, the language command is a Transact-SQL select command.

Note The *CTPUBLIC include* file is required in all source files that contain calls to Open ClientConnect.

Setting up the Client-Library programming environment

cs_ctx_alloc allocates a context structure. A context structure is used to store configuration parameters that describe a particular "context," or operating environment, for a set of connections.

Application properties that can be defined at the context level include the version of Client-Library being used, the login time-out value, and the maximum number of connections allowed within the context.

ct_init initializes your environment. It must be the first call in an application after cs_ctx_alloc.

Connecting to a server

ct_con_alloc allocates a connection structure. A connection structure contains information about a particular client/server connection.

ct_con_props sets and retrieves the property values of a connection. Connection properties include:

- User name and password, which are used in logging into a server
- Application name, which appears in Adaptive Server's sysprocesses table
- Packet size, which determines the size of network packets that an application sends and receives
- Dynamic network driver (LU 6.2, IBM TCP/IP, Interlink TCP/IP, CPI-C), which defines the type of the network used between Open ClientConnect and the server

Open ClientConnect includes a Connection Router where servers and server connections are defined.

For a complete list of connection properties, see "Properties" on page 37 of this book.

ct_connect opens a connection to a server, logging into the server with the connection information specified via ct_con_props.

Sending a command to the server

ct_cmd_alloc allocates a command structure. A command structure is used to send commands to a server and to process the results of those commands.

ct_command initiates the process of sending a command. In this example, it initiates a language command.

ct_send sends the command to the server.

Processing the results of the command

Almost all Client-Library programs process results by using a loop controlled by ct_results. Inside the loop, one of several actions takes place on the current type of result. Different types of results require different types of processing.

For row results, typically the number of columns in the result set is determined and then used to control a loop in which result items are bound to program variables. An application can call ct_res_info to get the number of result columns. After the result items are bound using ct_bind, the application calls to ct_fetch to fetch data rows until end-of-data.

The results-processing model used in the example looks like this:

- Retrieve results: ct_results.
- Determine the type of results by examining the value in the result_type field.
- Process results.
    - If results are result rows: ct_res_info, ct_describe, ct_bind, ct_fetch.
    - If results are return parameters: ct_param, ct_describe, ct_bind, ct_fetch.
    - If results are status: ct_bind, ct_fetch.

ct_results sets up results for processing. The ct_results return parameter result_type indicates the type of result data that is available for processing.

Note that the example program calls ct_results in a loop that continues as long as ct_results returns CS_SUCCEED, indicating that result sets are available for processing. Although this type of program structure is not strictly necessary in the case of a simple language command, it is highly recommended. In more complex programs, it is not possible to predict the number and type of result sets than an application will receive in response to a command.

ct_bind binds a result item to a program variable. Binding creates an association between a result item and a program data space.

ct_fetch fetches result data. In the example, since binding has been specified and the count field in the DATAFMT structure for each column is set to 1, each ct_fetch call copies one row of data into program data space. As each row is fetched, the example program prints it.

After the ct_fetch loop terminates, the example program checks its final return code to find out whether it dropped out because of end-of-data, or because of failure.

Finishing up

Use the following functions to close a connection and deallocate the structures:

- ct_cmd_drop deallocates a command structure.
- ct_close closes a server connection.
- ct_con_drop deallocates a connection structure.
- ct_exit cleans up the remaining resources being used by command or connection handles.
- cs_ctx_drop deallocates a context structure.

CHAPTER 2

Topics

The following topics are included in this chapter:

Buffers

Description

A number of arguments used in Client-Library functions affect the contents of buffers: *ACTION*, *BUFFER*, *BUFFER_LEN*, *BUFBLANKSTRIP*, and *OUTLEN*.

These arguments are described individually below. For a summary of argument values and their interaction, see Table 2-1 on page 23.

Arguments

ACTION describes what is done to the data. For most functions, *ACTION* can take the following symbolic values:

CS_SET	Assigns a value
CS_GET	Retrieves a value
CS_CLEAR	Clears the buffer, or, for functions that set properties, resets to the default property value

BUFFER is a program variable, called a “buffer.”

When *ACTION* is *CS_SET*	*BUFFER* is the data being read by the function.
When *ACTION* is *CS_GET*	*BUFFER* is the information retrieved by the function.
When *ACTION* is *CS_CLEAR*	*BUFFER* remains unchanged.

BUFFER_LEN is the length, in bytes, of the *BUFFER* argument.

When the value in the buffer is a fixed-length or symbolic value:	Assign *BUFFER_LEN* the actual length of the value or CS_UNUSED.
When the value in the buffer is of variable length:	Assign *BUFFER_LEN* the maximum number of bytes that the buffer can contain.

If *BUFFER_LEN* is set to CS_GET and the buffer is too small to hold the returned value, the function returns CS_FAIL in the *RETCODE* argument and the actual length of the requested information in *OUTLEN*.
When this happens, you can query the length of the incoming data by setting CS_SET to 0 and executing the function. The actual length of the data is returned to the *OUTLEN* argument. Enlarge the buffer, assign the value in *OUTLEN* to *BUFFER_LEN*, and execute the function again.
BUFBLANKSTRIP is the blank stripping indicator. It indicates whether or not trailing blanks are stripped. This argument gets one of the following symbolic values:

CS_TRUE	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character it contains.
CS_FALSE	Trailing blanks are not stripped; they are included in the value.

OUTLEN is an integer variable where the function returns the actual length, in bytes, of the data being retrieved during a CS_GET operation. For all other operations, *OUTLEN* is ignored.
When the retrieved data is longer than *BUFFER_LEN* bytes, an application can use the value of *OUTLEN* to determine how many bytes are needed to hold the information.

To do this, set the value of *OUTLEN* to 0 and call the function. Then set the value of *BUFFER_LEN* to the length returned in *OUTLEN* and call the function again.

Summary of buffer entries

Table 2-1 summarizes the interaction between *ACTION*, *BUFFER*, *BUFFER_LEN*, and *OUTLEN*.

Table 2-1: ACTION, BUFFER, BUFFER_LEN, and OUTLEN

For this ACTION	When BUFFER	BUFFER_LEN is	OUTLEN is	Result
CS_CLEAR	N/A	CS_UNUSED	Ignored	The property reverts to its default value.
CS_SET	Contains a variable-length character string	The length of the string	Ignored	The data in *BUFFER* is read by the function.
CS_SET	Contains a variable-length character string for which the terminating character is the last non-blank character	The maximum length of the string	Ignored	The application sets the value of *BUFBLANKSTRIP* to CS_TRUE. The data in *BUFFER* is read by the function.
CS_SET	Contains a fixed-length character string or symbolic value	CS_UNUSED	Ignored	The data in *BUFFER* is read by the function.
CS_GET	Is large enough for the return character string	The length of the buffer	Set by Client-Library	The retrieved value is copied to the buffer. *OUTLEN* returns the length of the retrieved value.
CS_GET	Is not large enough for the return character string	The length of the buffer	Set by Client-Library	No data is copied to the buffer. *OUTLEN* returns the length of the value being retrieved. The function returns CS_FAIL, indicating that you should assign the value returned here to the length argument and execute the program again.
CS_GET	Is assumed to be large enough for a fixed-length or symbolic value	CS_UNUSED	Set by Client-Library	The retrieved value is copied to the buffer. *OUTLEN* returns the length of the value being retrieved.

CLIENTMSG structure

Description

A CLIENTMSG (client message) structure contains information about an error or informational message returned by Open ClientConnect. This structure is defined within the application. When the error involves interaction with the operating system, the operating system error information is returned to this structure. CTBDIAG returns a message string and information about the message into CLIENTMSG.

Server messages are returned to a SERVERMSG structure, described in "SERVERMSG structure" on page 48. A sample CLIENTMSG structure is provided in the CTPUBLIC copybook.

Definition

A CLIENTMSG structure is defined as follows:

```
DCL
01 CLIENTMSG,
05 CMSG_SEVERITY              FIXED BIN(31),
05 CMSG_OC_MSGNO              FIXED BIN(31),
05 CMSG_OC_MSGTEXT            CHAR(256),
05 CMSG_OC_MSGTEXT_LEN        FIXED BIN(31),
05   CMSG_OS_MSGNO            FIXED BIN(31),
05 CMSG_OS_MSGTEXT            CHAR(256),
05 CMSG_OS_MSGTEXT_LEN        FIXED BIN(31),
05 CMSG_STATUS                  FIXED BIN(31);
```

- *CMSG_SEVERITY* is a symbolic value representing the severity of the message. Severity values are provided in the CTPUBLIC copybook.

  Table 2-2 lists the legal values for *CMSG_SEVERITY*.

Table 2-2: Legal Values for CMSG SEVERITY

CMSG_SEVERITY value	Meaning
CS_SV_INFORM (0)	No error occurred. The message is informational.
CS_SV_API_FAIL (1)	A Client-Library routine generated an error. This error is typically caused by a bad parameter or calling sequence. The server connection is probably salvageable.
CS_SV_RETRY_FAIL (2)	An operation failed, but the operation can be retried.
CS_SV_RESOURCE_FAIL (3)	A resource error occurred. This error is typically caused by an allocation error, a lack of file descriptors, or timeout error. The server connection is probably not salvageable.
CS_SV_CONFIG_FAIL (4)	A configuration error occurred.
CS_SV_COMM_FAIL (5)	An unrecoverable error in the server communication channel occurred. The server connection is not salvageable.
CS_SV_INTERNAL_FAIL (6)	An internal Client-Library error occurred.
CS_SV_FATAL (7)	A serious error occurred. All server connections are unusable.

- *CMSG_OC_MSGNO* is the Client-Library message number. Client-Library messages are listed in the Mainframe Connect Client Option *Installation and Administration Guide*.
- *CMSG_OC_MSGTEXT* is the text of the Client-Library message string.
- *CMSG_OC_MSGTEXT_LEN* is the length, in bytes, of *CMSG_OC_MSGTEXT*. If there is no message text, the value of *CMSG_OC_MSGTEXT_LEN* is 0.
- *CMSG_OS_MSGNO* is the server error number, if any. A value here indicates that the message involved CICS or IMS TM I/O errors, remote server errors, or Transaction Router Service (TRS) errors.
- *CMSG_OS_MSGTEXT* is the text of the operating system message string, if any.
- *CMSG_OS_MSGTEXT_LEN* is the length, in bytes, of *CMSG_OS_MSGTEXT*. If there is no message text, the value of *CMSG_OS_MSGTEXT_LEN* is 0.
- *CMSG_STATUS* is reserved for future use.

Customization

Description

When installing Open ClientConnect, system programmers customize the product for the customer site, defining language and program characteristics locally. Some of the customized items are used by Gateway-Library programs.

Locally-defined items

The following locally-defined items are used by Gateway-Library functions:

- An access code, which is required to retrieve a client password.

  Two customization options are related to the ability to retrieve client passwords:

  - The access code itself is defined during customization.
  - An access code flag is set to indicate whether the access code is required to retrieve the client password
- The native language used at the mainframe (The default is US-English).
- Whether DB2 LONG VARCHAR data strings with lengths that are greater than 255 bytes are truncated or rejected when sent to a client.

Customization instructions are in the Mainframe Connect Server Option *Installation and Administration Guide*. The customization module is loaded during program initialization (CTBINIT).

DATAFMT structure

Description

A DATAFMT structure is used to describe data values and program variables. For example:

- CTBBIND requires a DATAFMT structure describing a destination variable.
- CTBDESCRIBE returns a DATAFMT structure describing a result data item.
- CTBPARAM requires a DATAFMT structure describing an input parameter.
- CSBCONVERT requires a DATAFMT structure describing source and destination data.

Most functions use only a subset of the fields in a DATAFMT structure. For example, CTBBIND does not use the FMT_NAME, FMT_STATUS, and FMT_UTYPE fields, and CTBDESCRIBE does not use the FMT_FORMAT field. For information on which DATAFMT fields a function uses, see Table 2-3 on page 27 in this chapter, or the description of the relevant function in the next chapter.

Definition

A DATAFMT structure is defined as follows:

```
DCL
01 DATAFMT,
05 FMT_NAME       CHAR(132),
05 FMT_NAMELEN    FIXED BIN(31),
05 FMT_TYPE       FIXED BIN(31),
05 FMT_FORMAT     FIXED BIN(31),
05 FMT_MAXLEN     FIXED BIN(31),
05 FMT_SCALE      FIXED BIN(31),
05 FMT_PRECIS     FIXED BIN(31),
05 FMT_STATUS     FIXED BIN(31),
05 FMT_COUNT      FIXED BIN(31),
05 FMT_UTYPE      FIXED BIN(31),
05 FMT_LOCALE     FIXED BIN(31);
```

Table 2-3 describes the fields in the DATAFMT structure.

Table 2-3: Fields in the DATAFMT structure

Field	Contents	Used by
FMT_NAME	The name of the data item.	CTBDESCRIBE CTBPARAM
FMT_NAMELEN	The length of FMT_NAME.	CTBDESCRIBE CTBPARAM
FMT_TYPE	The datatype of the data. See the specific call to find which data this refers to.	CSBCONVERT CTBBIND CTBDESCRIBE CTBPARAM
FMT_FORMAT	The format of the data, represented by symbolic values.	CTBBIND
FMT_MAXLEN	The maximum length of the data.	CSBCONVERT CTBBIND CTBDESCRIBE CTBPARAM
FMT_SCALE	The number of digits in the decimal part of a number. This field is used with packed decimal, numeric and Sybase-decimal.	CSBCONVERT CTBBIND
FMT_PRECIS	The total number of digits in a number. This field is used with packed decimal, numeric and Sybase-decimal.	CSBCONVERT CTBBIND

Field	Contents	Used by
FMT_STATUS	Status values.	CTBDESCRIBE CTBPARAM
FMT_COUNT	The number of items.	CTBBIND CTBDESCRIBE
FMT_UTYPE	The user-defined datatype (UDT) of retrieved data. The UDT is assigned by the server.	CTBDESCRIBE CTBPARAM
FMT_LOCALE	Reserved for future use.	CSBCONVERT CTBBIND CTBDESCRIBE CTBPARAM

- FMT_NAME is the name of the data item. This can be a column, a parameter, or a return status name.
- FMT_NAMELEN is the length, in bytes, of FMT_NAME. Assign FMT_NAMELEN a value of 0 if the data item is unnamed.
- FMT_TYPE is the datatype of the data. This is one of the Client-Library datatypes listed under "Datatypes" on page 30.

Note Return status values have a datatype of CS_INT.

- FMT_FORMAT is the destination format of fixed-length character or binary data. FMT_FORMAT is one of the following values listed in Table 2-4.

Table 2-4: Values for the DATAFMT field FMT_FORMAT

Value	Meaning	Datatype
CS_FMT_PADBLANK	The data should be padded with blanks to the full length of the destination variable.	For fixed-length character data only.
CS_FMT_PADNULL	The data should be padded with zeroes to the full length of the destination variable.	For binary or fixed-length character data.

- FMT_MAXLEN can represent various lengths, depending on which function is using the DATAFMT structure. Lengths are represented in bytes. Table 2-5 lists the meaning of FMT_MAXLEN for each function that uses it.

Table 2-5: Lengths defined by the DATAFMT field FMT_MAXLEN

Function	Length defined
CSBCONVERT	The length of the source variable and the length of the destination variable.
CTBBIND	The length of the variable to which the data is bound.

Function	Length defined
CTBDESCRIBE	The maximum possible length of the column or parameter being described.
CTBPARAM	The maximum length of return parameter data.

- FMT_SCALE is the number of decimal places in the value being converted. FMT_SCALE is used with CTBBIND, CSBCONVERT, and CTBPARAM when converting to or from decimal datatypes CS_PACKED370, numeric, and Sybase-decimal.

  Legal values for FMT_SCALE are from 0 to 31. If the actual scale is greater than 31, the call fails.

  To indicate that destination data should use the same scale as the source data, set FMT_SCALE to CS_SRC_VALUE. FMT_SCALE must be less than or equal to FMT_PRECIS.

- FMT_PRECIS is the precision of the value being converted. FMT_PRECIS is used only with CTBBIND, CSBCONVERT, and CTBPARAM when converting to or from decimal datatypes CS_PACKED370, numeric, and Sybase-decimal. Legal values for FMT_PRECIS are from 1 to 31. To indicate that destination data should use the same precision as the source data, set FMT_PRECIS to CS_SRC_VALUE. The value of FMT_PRECIS must be greater than or equal to FMT_SCALE.

- FMT_STATUS is one or more of the following symbolic values (added together) listed in Table 2-6.

Table 2-6: Values for the DATAFMT field FMT_STATUS

Value	Meaning	For this function
CS_CANBENULL	The column can contain nulls.	CTBDESCRIBE
CS_NODATA	No data is associated with the result data item.	CTBDESCRIBE
CS_INPUTVALUE	The parameter is a non-return RPC parameter (input parameter).	CTBPARAM
CS_RETURN	The parameter is an RPC return parameter.	CTBPARAM

- FMT_COUNT is the number of rows to copy to program variables per CTBFETCH call.

- FMT_UTYPE is the user-defined datatype, if any, of data returned by the server.

  Note FMT_UTYPE is used only for datatypes defined at the server, not for datatypes defined by Open ClientConnect. For example, Adaptive Server-defined datatypes or datatypes defined with TDSETUDT in Open ServerConnect.

- FMT_LOCALE is reserved for future use. It must be set to zero.

Datatypes

Description

Open ClientConnect supports a wide range of datatypes. These datatypes are shared with Open Client, Open Server and Open ServerConnect, and correspond directly to Adaptive Server datatypes.

Table 2-7 lists the Client-Library datatypes, together with the corresponding type constants, Adaptive Server datatypes, and Open ServerConnect datatypes.

Table 2-7: Summary of Open ClientConnect datatypes

This Client-Library datatype	Whose datatype declaration looks like this	Describes this type of data	Corresponds to this Adaptive Server datatype	Corresponds to this Open ServerConnect datatype
CS_BINARY	CHAR(n)	Binary	Binary	TDSBINARY
CS_CHAR	CHAR(n)	Character	Char	TDSCHAR
CS_DATETIME	CHAR(8)	8-byte datetime	Datetime	TDSDATETIME
CS_DATETIME4	CHAR(4)	4-byte datetime	Smalltime	TDSDATETIME4
CS_FLOAT	FLOAT DEC(n)	8-byte float	Float	TDSFLT8
CS_INT	FIXED BIN(31)	4-byte integer	int	TDSINT4
CS_LONGBINARY	CHAR (n)	Long variable binary	--	TDSLONGVARBIN
CS_LONGCHAR	CHAR (n)	Long variable character	--	TDSLONGVARCHAR
CS_MONEY	FIXED DEC(p,s)	8-byte money	money	TDSMONEY
CS_MONEY4	FIXED DEC(p,s)	4-byte money	smallmoney	TDSMONEY4
CS_PACKED370	FIXED DEC(p,s)	IBM S/370 packed decimal	decimal	TDS_PACKED_ DECIMAL

This Client-Library datatype	Whose datatype declaration looks like this	Describes this type of data	Corresponds to this Adaptive Server datatype	Corresponds to this Open ServerConnect datatype
CS_REAL	FLOAT DEC()	4-byte float	real	TDSFLT4
CS_SMALLINT	FIXED BIN(15)	2-byte integer	smallint	TDSINT2
CS_VARBINARY	BIT(n) VAR	Variable-length binary	--	TDSVARYBIN
CS_VARCHAR	CHAR(n) VAR	Variable-length character	--	TDSVARYCHAR
CS_NUMERIC	CHAR (35)	--	numeric	TDSNUMERIC
CS_DECIMAL	CHAR (35)	--	Sybase decimal	TDS_SYBASE_DECIMAL

Open ClientConnect datatypes

Open ClientConnect datatypes are designed to match the corresponding DB2 datatypes. For example, CS_VARCHAR is the same as the DB2 datatype VARCHAR, which is different from the Adaptive Server VARCHAR.

The following subsections describe each Open ClientConnect datatype in detail:

- Binary
- Character
- Datetime
- Integer
- Real, float, packed decimal, numeric and Sybase-decimal
- Money

Binary

Open ClientConnect supports the following three binary types:

- CS_BINARY is a binary type.
- CS_VARBINARY is a variable-length binary type. It corresponds to the DB2 datatype VARBINARY, the DB-Library datatype DBVARYBIN, and the Gateway-Library datatype TDSVARYBIN, and includes a length specification (the initial two bytes, referred to in print as “LL”) along with the data.

- CS_LONGBINARY is a long variable binary type. It does not include the two-byte “LL” length specification prefix. The default maximum length for this datatype is 32K.

Note CS-VARBINARY does not correspond to the Adaptive Server datatype varbinary. Open ClientConnect converts Adaptive Server varbinary data to CS-VARBINARY.

Character

Open ClientConnect supports the following three character types:

- CS_CHAR is a set-length character type.
- CS_VARCHAR is a variable-length character type. It corresponds to the DB2 datatype VARCHAR, the DB-Library datatype DBVARYCHAR, and the Gateway-Library datatype TDSVARYCHAR, and includes a length specification (the initial two bytes, referred to in print as “LL”) along with the data.
- CS_LONGCHAR is a long variable-length character type. It does not include the two-byte “LL” length specification prefix.

Note CS-VARCHAR does not correspond to the Adaptive Server datatype varchar. Open ClientConnect converts Adaptive Server varchar data to CS-VARCHAR.

Datetime

Open ClientConnect supports the following two datetime types that hold 8-byte and 4-byte datetime values, respectively:

- CS_DATETIME corresponds to the Adaptive Server datatype. The range of legal CS-DATETIME values is from January 1, 1753 to December 31, 9999, with a precision of 1/300th of a second (3.33 milliseconds).
- CS_DATETIME4 corresponds to the Adaptive Server datatype. The range of legal CS_DATETIME4 values is from January 1, 1900 to June 6, 2079, with a precision of 1 minute.

Integer

Open ClientConnect supports the following two integer types:

- CS_SMALLINT is a 2-byte integer.
- CS_INT is a 4-byte integer.

Real, float, packed decimal, numeric and Sybase-decimal

Open ClientConnect supports the following five decimal types:

- CS_REAL is a 4-byte decimal value.
- CS_FLOAT is an 8-byte decimal value.

- CS_PACKED370 is used to handle IBM S/370 packed decimal data. It can be converted to the Adaptive Server numeric, or Sybase-decimal datatype with CSBCONVERT.
- CS_PACKED370 values can be negative. The maximum number of decimal places for a packed decimal object is 31.
- CS_NUMERIC is used to handle Adaptive Server numeric data. It can be converted to character or packed decimal, as in the following example:

```
DCL
      01 NUMDEC,
      05 Precision CHAR(1),
      05 Scale     CHAR(1),
      05 APR       CHAR(33)
```

- CS_DECIMAL is used to handle Adaptive Server numeric data. It can be converted to character or packed decimal. It is defined the same way as CS_NUMERIC.

Money

Open ClientConnect supports the following two money datatypes that hold 8-byte and 4-byte money values, respectively:

- CS_MONEY corresponds to the Adaptive Server datatype. The range of legal CS_MONEY values is +/- $922,337,203,685,477.5807.
- CS_MONEY4 corresponds to the Adaptive Server datatype. The range of legal CS_MONEY4 values is between -$214,748.3648 and +$214,748.3647.

Error and message handling

Description

All Open ClientConnect functions return success or failure indications in their *RETCODE* arguments. It is highly recommended that applications check these return codes for each call.

In addition, Open ClientConnect applications must handle three types of error and informational messages:

- Open ClientConnect messages, also known as ‘‘client messages,’’ are generated by the functions documented in this book. They range in severity from informational messages to fatal errors.
- Operating system messages.

- Server messages are generated by the server. They range in severity from informational messages to fatal errors.

For a list of messages, see the Mainframe Connect Client Option *Installation and Administration Guide*.

Return codes

Client-Library return codes begin with “CS_”. The codes returned to each Client-Library function are listed on the reference pages for that function, under “Return value.”

Gateway-Library return codes all begin with “TDS_”. However, on the mainframe, a few TDS_XXX return codes are returned to Client-Library functions. Some are returned to both Gateway-Library and Client-Library functions.

TDS_XXX return codes that are returned to specific functions only are documented on the reference pages for those functions, under “Return value.” Programs using these functions should check for these codes.

Some TDS_XXX return codes can be returned to any function under certain circumstances.

You can find a list of all TDS_XXX codes in the Mainframe Connect Client Option *Installation and Administration Guide*.

Messages

A Client-Library application uses the function CTBDIAG to handle messages in line. CTBDIAG returns message information to two structures defined within the application: the CLIENTMSG structure for client messages, and the SERVERMSG structure for server messages.

An application calls CTBDIAG to initialize in-line message handling for a connection. CTBDIAG cannot be used at the context level.

Note Whenever a Client-Library function returns CS_FAIL, you must run CTBDIAG to determine what the error is.

An application can retrieve messages into SQLCA and SQLCODE structures. If the application uses these structures, it must set the property CS_EXTRA_INF to CS_TRUE, using CTBCONALLOC. This is because the SQL structures require information that Client-Library does not customarily return. If CS_EXTRA_INF is not set, a loss of information occurs. For more information, see “The CS_EXTRA_INF property” on page 35, and “SQLCA structure” on page 50.

For additional information on the in-line method of handling function and server messages, see the reference page for the function CTBDIAG on page 119, as well as the topics "CLIENTMSG structure" on page 24 and "SERVERMSG structure" on page 48.

The CS_EXTRA_INF property

The CS_EXTRA_INF property is used by an application to determine the number of rows affected by the most recent command. If you want this extra information, you must set this property before you call the function.

An application can determine the number of rows in two ways:

- An application that is retrieving messages into a SQLCA or SQLCODE structure must set the property CS_EXTRA_INF to CS_TRUE, using CTBCONPROPS. This is necessary because the SQLCA structure needs to know the number of rows affected, information that functions do not customarily return. If CS_EXTRA_INF is not set, a loss of information occurs.
- An application that is not using SQLCA or SQLCODE can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard Client-Library messages.

For further information on the use of the SQLCA and SQLCODE structures in Open ClientConnect, turn to "SQLCA structure" on page 50 and "SQLCODE structure" on page 52.

For three-tier processing, requests directed to an Adaptive Server or an Open Server application are sent to Mainframe ClientConnect (MCC), supplied with the Sybase DirectConnect product.

For two-tier processing, requests are sent directly to an Adaptive Server.

MCC allows mainframe transactions to access the LAN-based environment in which Sybase servers operate. It logs in to the target server, passing along login information, does the necessary conversions, and forwards the request. When results are ready, it passes them from the server to the client, again performing any necessary conversions.

MCC is transparent to the mainframe application. You specify the name of the server in the *SERVERNAME* parameter of the CTBCONNECT call, and the MCC forwards the request to the specified server.

To learn how CICS or IMS TM determines the MCC and connections to use to connect to the target server, see the Mainframe Connect Server Option *Installation and Administration Guide*. The client program is not concerned with these details.

Note Requests to Open ServerConnect are sent directly from one transaction processing station to another and do not use Mainframe ClientConnect.

Nulls

Description

Client-Library allows parameters to have a “null” value — that is, to contain no information, not even blanks. In PL/1, a null value is usually represented by zeroes.

NULL and unused parameters

There are several rules for assigning null values to arguments:

- For *handles*, an application assigns the following symbolic values to indicate a null:
  - CS_NULL_CONTEXT for context handles.
  - CS_NULL_CONHANDLE for connection handles.
  - CS_NULL_CMD for command handles.
- For *output arguments*:
  - Arguments that return a single integer are never null.
  - Arguments that return longer values can be null. A null value is indicated by a separate argument, indicating that the argument of interest should be treated as null.
- For *arguments that have a corresponding length argument*, assign the value CS_NULL_STRING to the corresponding length argument, if one is present, to indicate that the value of an argument should be treated as null.

  For *DATAFMT structures*, you indicate a null field by setting FMT_MAXLEN to zero.
- For *arguments that have NULL indicators*, assign CS_PARAM_NULL to the indicator argument.

For example, CTBBIND has the arguments *COPIED* and *COPIED_NULL*. If *COPIED* is null, assign CS_PARAM_NULL to *COPIED_NULL*. If *COPIED* is not null, assign CS_PARAM_NOTNULL to *COPIED_NULL*.

- For *all other variables*, assign CS_UNUSED to indicate that the argument should be ignored.

Padding with NULLs

The FMT_FORMAT field of the DATAFMT structure of CTBBIND can be padded with either blanks or nulls. CS_FM_PADNULL pads with zeroes; CS_FM_PADBLANK pads with blanks.

Properties

Description

Properties define aspects of Open ClientConnect behavior.

Login properties are used when logging into a server. Login properties include CS_USERNAME, CS_PASSWORD, CS_PACKETSIZE, and CS_NET_DRIVER (used with dynamic network drivers).

A server can change the values of some login properties during the login process. For example, if an application sets CS_PACKETSIZE to 2048 bytes and then logs into a server that cannot support this packet size, the server overwrites 2048 with a packet size it can support. These types of properties are called *negotiated properties*.

CS_NET_DRIVER is used to switch communication protocols. The supported protocols vary depending on the operating environment. TCP/IP and CPIC are supported in CICS, IMS and MVS. SNA (LU 6.2) is only supported in CICS and IMS.

Setting and retrieving properties

An application calls CTBCONFIG, CTBCONPROPS, and CTBCMDPROPS to set and retrieve properties at the context, connection, and command structure levels, respectively. An application calls CTBCONFIG to set and retrieve most context properties; it calls CSBCONFIG to set and retrieve global context properties.

When a context structure is allocated, its property values default to standard values.

When a connection structure is allocated, it picks up default property values from its parent context. For example, if CS_TEXTLIMIT is set to 16,000 at the context level, then any connection created within this context has a default text limit value of 16,000. Likewise, when a command structure is allocated, it picks up default property values from its parent connection.

An application can override a default property value by calling CSBCONFIG, CTBCONFIG, CTBCONPROPS, or CTBCMDPROPS to change the value of the property.

Most property values can be either set or retrieved by an application, but some properties are "retrieve only."

Summary of properties

Table 2-8 lists the Open ClientConnect properties.

Table 2-8: Open ClientConnect properties

Property	Meaning	Values	Function set by	Notes
CS_APPNAME	The application name used when logging into the server.	A character string. The default is NULL.	CTBCONPROPS	Login property. Takes effect only if set before the connection is established.
CS_CHARSETCNV	The conversion indicator. It indicates whether or not character set conversion is taking place.	CS_TRUE or CS_FALSE. A default is not applicable.	CTBCONPROPS	Retrieve only, after the connection is established.
CS_COMMBLOCK	A pointer to a communication sessions block (EIB).	A pointer value. The default is NULL.	CTBCONPROPS	Takes effect only if set before the connection is established.
CS_EXTRA_INF	The extra information indicator. It specifies whether or not to return the extra information that is required when processing messages in-line using a SQLCA or SQLCODE.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	CSBCONFIG, CTBCONFIG, CTBCONPROPS	

Property	Meaning	Values	Function set by	Notes
CS_HOSTNAME	The host (server) machine name.	A character string. The default is NULL.	CTBCONPROPS	Login property. Takes effect only if set before the connection is established.
CS_LOC_PROP	A pointer to a CS_LOCALE structure that defines localization information.	A pointer value. A connection picks up a default CS_LOC_PROP from its parent context.	CTBCONPROPS	Login property.
CS_LOGIN_STATUS	The connection status indicator. It indicates whether or not the connection is open.	CS_TRUE or CS_FALSE. A default is not applicable.	CTBCONPROPS	Retrieve only.
CS_LOGIN_TIMEOUT	The login timeout value.	An integer value. The default is 60 seconds. A value of CS_NO_LIMIT represents an infinite timeout period.	CTBCONFIG	Open ClientConnect ignores this property. This is the same value as the CICS RTIMEOUT.
CS_MAX_CONNECT	The maximum number of connections for this context.	An integer value. The default varies by platform. On mainframes, the default is 25 (an unlimited number of connections can be defined for a context).	CTBCONFIG	
CS_NET_DRIVER	The type of network driver in use.	CS_LU62, CS_TCPIP, CS_INTERLINK, or CS_NCPIC. Defaults for: CICS – CS_LU62 IMS – CS_LU62 MVS – CS_NCPIC	CTBCONPROPS	

Property	Meaning	Values	Function set by	Notes
CS_NETIO	The sync/async indicator. It indicates whether network I/O is synchronous or asynchronous.	CS_SYNC_IO or CS_ASYNC_IO. The default is CS_SYNC_IO.	CTBCONFIG, CTBCONPROPS	With Open ServerConnect this value is always CS_SYNC_IO.
CS_NOINTERRUPT	The interrupt indicator. It indicates whether or not the application can be interrupted.	CS_TRUE or CS_FALSE. The default is CS_FALSE, which means the application can be interrupted.	CTBCONFIG, CTBCONPROPS	N/A for CICS. This property is included for compatibility with other Open Client libraries.
CS_PACKETSIZE	The TDS packet size.	An integer value. The default varies by platform. On UNIX and MVS platforms, the default is 512 bytes.	CTBCONPROPS	Negotiated login property. Takes effect only if set before the connection is established.
CS_PASSWORD	The password used to log in to the server.	A character string. The default is NULL.	CTBCONPROPS	Login property. Takes effect only if set before the connection is established.
CS_TDS_VERSION	The version of the TDS protocol that the connection is using.	A symbolic version level. CS_TDS_VERSION defaults to the value of CS_VERSION.	CTBCONPROPS	Negotiated login property. Takes effect only if set before the connection is established.
CS_TEXTLIMIT	The largest text or image value to be returned on this connection.	An integer value. The default is CS_NO_LIMIT.	CTBCONFIG, CTBCONPROPS	
CS_TIMEOUT	The timeout value.	An integer value. The default is CS_NO_LIMIT.	CTBCONFIG	Not supported under CICS. CICS waits for ever.
CS_TRANSACTION_ NAME	A transaction name.	A string value. The default is NULL.	CTBCONPROPS	

Property	Meaning	Values	Function set by	Notes
CS_USERDATA	User-allocated data.	User-allocated data.	CTBCONPROPS, CTBCMDPROPS	These are pointers to data that allow the customer to tie in to the data.
CS_USERNAME	The name used to log in to the server.	A character string. The default is NULL.	CTBCONPROPS	Login property. Takes effect only if set before the connection is established.
CS_VERSION	The version of Client-Library used by this context.	CS_VERSION gets its value from a context CTBINIT call.	CSBCONFIG, CTBCONFIG	

About the properties

Application name

- CS_APPNAME defines the application name that a connection uses when connecting to a server.

Character set conversion

- CS_CHARSETCNV indicates whether or not the server is converting between the client and server character sets. This property is retrieve-only, after a connection is established.

  A value of CS_TRUE indicates that the server is converting between the client and server character sets; CS_FALSE indicates that no conversion is taking place.

Communications session block

- CS_COMMBLOCK defines a pointer to a communications block (EIB).

Extra information

- CS_EXTRA_INF determines whether or not Open ClientConnect returns the extra information that CTBDIAG requires to fill in SQLCA or SQLCODE structures.

  This extra information includes the number of rows affected by the most recent command.

  If an application is not retrieving messages into a SQLCA or SQLCODE, the extra information is returned as ordinary Client-Library messages.

Host name

- CS_HOSTNAME is the name of the host machine, used when logging in to a server.

Locale information

- CS_LOC_PROP defines a pointer to a CS_LOCALE structure, which contains localization information. Localization information includes a language, a character set, datetime, money, and numeric formats, and a collating sequence. This property must be set to 0.

Login status

- CS_LOGIN_STATUS is CS_TRUE if a connection is open, CS_FALSE if it is not. This property can only be retrieved.
- CTBCONNECT is used to open a connection.

Login timeout

- CS_LOGIN_TIMEOUT defines the length of time, in seconds, that an application waits for a login response when making a connection attempt. Timeouts are not supported under CICS.

Maximum number of connections

- CS_MAX_CONNECT defines the maximum number of simultaneously open connections that a context can have. The default varies by platform. Negative and zero values are not allowed for CS_MAX_CONNECT.
  - On mainframes, CS_MAX_CONNECT has a default value of 25 (an unlimited number of connections can be defined for a context).
  - If CTBCONFIG is called to set a value for CS_MAX_CONNECT that is less than the number of currently open connections, CTBCONFIG generates an error and returns CS_FAIL without altering the value of CS_MAX_CONNECT.

Network driver

- CS_NET_DRIVER determines the type of dynamic network driver that is used. Possible values are:
  - CS_INTERLINK
  - CS_LU62
  - CS_NCPIC
  - CS_TCPIP

  The default value for CICS and IMS is CS_LU62. The default value for MVS is CS_NCPIC.

Network I/O

- CS_NETIO determines whether a connection is synchronous or asynchronous.

  Because Open ClientConnect does not support asynchronous processing, this value is always CS_SYNC_IO.

No interrupt

- CS_NOINTERRUPT is not supported for Open ClientConnect. It is included for compatibility with other Open Client libraries.

Packet size

- CS_PACKETSIZE determines the packet size that Open ClientConnect uses when sending Tabular Data Stream (TDS) packets.
- If an application needs to send or receive large amounts of text, image, or bulk data, a larger packet size can improve efficiency. The default packet size is 512.

Password

- CS_PASSWORD defines the password that a connection uses when logging in to a server.

TDS version

- CS_TDS_VERSION defines the version of the Tabular Data Stream (TDS) protocol that the connection is using.

  Because CS_TDS_VERSION is a negotiated login property, its value can change during the login process. An application can set CS_TDS_VERSION to request a TDS level before calling CTBCONNECT. In this case, when CTBCONNECT creates the connection, it looks for the requested TDS version. If the server cannot provide the requested TDS version, a new (lower) TDS version is negotiated.
  An application can retrieve the value of CS_TDS_VERSION after a connection is established to determine the actual version of TDS in use.

  The following table lists the symbolic values that CS_TDS_VERSION can have:

Value	Indicates	Features supported
CS_TDS_46	4.6 TDS	Registered procedures, TDS passthrough, negotiable TDS packet size, multibyte character sets.
CS_TDS_50	5.0 TDS	Accesses system Adaptive Server 10.0 and above. **Note** TDS 5.0 only works with an Adaptive Server 10.0 and above.

Text and image limit

- CS_TEXTLIMIT indicates the length, in bytes, of the longest text or image value that an application wants to receive. Open ClientConnect reads but ignores any part of a text or image value that goes over this limit.

  The default value of CS_TEXTLIMIT is CS_NO_LIMIT, meaning the application reads and returns all data sent by the server.

Timeout

- CS_TIMEOUT controls the length of time, in seconds, that Client-Library waits for a server response when making a request.

  This value is ignored by Open ClientConnect.

Transaction name

- CS_TRANSACTION_NAME names a transaction. If the accessed server is a Gateway-Library application, this is the name of the transaction.
- Calls to Adaptive Server do not require a transaction name.
- All Client-Library applications can set CS_TRANSACTION_NAME. If a transaction name is not required, CS_TRANSACTION_NAME is ignored.

User data

- CS_USERDATA property defines user-allocated data. This property allows an application to associate user data with a particular connection or command structure. An application allocates a data space from which it can get this data when needed.
- To associate user data with a context structure, an application calls CSBCONFIG.
- A PL/I program can use the Working Storage section to define this data.

User name

- CS_USERNAME defines the user login name that the connection uses to log into a server.

Version of Open ClientConnect

- CS_VERSION represents the version of Open ClientConnect behavior that an application requests through CTBINIT. The value of this property can only be retrieved.
- Connections allocated within a context pick up default CS_TDS_VERSION values from their parent context CS_VERSION level.

Remote procedure calls (RPCs)

Description

A client application can call a stored procedure on an Adaptive Server or an Open ServerConnect transaction running in a separate CICS or IMS region.

A client application can call a stored procedure or mainframe transaction one of two ways:

- By executing an Adaptive language request (for example, “execute myproc”)
- By making an RPC

Comparing RPCs and execute statements

RPCs have a few advantages over execute statements:

- An RPC can be used to execute an Adaptive Server stored procedure or any Open ServerConnect transaction.
- An Adaptive language request can only be used to execute an Adaptive Server stored procedure or a specially written Open ServerConnect language transaction.
- When sending a request to an Adaptive Server, it is simpler and faster to accommodate stored procedure return parameters if the procedure is invoked with an RPC instead of a language request.

Servers executing remote procedures

A server can execute a stored procedure or transaction residing on another server. This might occur when a stored procedure being executed on one Adaptive Server contains an execute statement for a stored procedure on another Adaptive Server. The execute command causes the first server to log in to the second server and execute the remote procedure. This is called a server-to-server RPC. It happens without any intervention from the application, although the application can specify the remote password that the first server uses to log in to the second.

A server-to-server RPC also occurs when an application sends a request to execute a stored procedure that does not reside on the server to which it is directly connected.

Note SQL commands contained in a stored procedure that is executed as the result of a server-to-server remote procedure call cannot be rolled back.

RPC routines

The following functions relate to RPCs:

- CTBREMOTEPWD sets and clears the passwords that are used when logging in to a remote server (This feature is not available for calls to Open ServerConnect).
- CTBCOMMAND initiates an RPC.
- CTBPARAM defines parameters for an RPC.
- CTBSEND sends an RPC.
- CTBRESULTS, CTBBIND, and CTBFETCH process remote procedure results.

RPC results

In addition to results generated by the SQL statements they contain, Adaptive Server stored procedures and Open ServerConnect transactions that are executed through an RPC:

- Can generate a return parameter result set

- Always generate a return status result set

All types of results—rows, status, and parameters—can be processed using CTBRESULTS, CTBBIND, and CTBFETCH.

Stored procedure return parameters

Adaptive Server stored procedures and mainframe server transactions can return values for specified return parameters. Changes made to the value of a return parameter inside the stored procedure or transaction are then available to the program that called the procedure or transaction. This is analogous to the "pass by reference" facility available in some programming languages.

In order for a parameter to function as a return parameter, it must be declared as such within the stored procedure. Client-Library applications use the CTBPARAM routine to indicate return parameters.

Processing RPC return parameters

Return parameter values are available to an application as a parameter result set only if the application invoked the stored procedure using an RPC.

CTBRESULTS returns CS_PARAM_RESULT if a parameter result set is available to be processed. Because stored procedure parameters are returned to an application as a single row, one call to CTBFETCH copies all of the return parameters for a stored procedure into the program variables designated through CTBBIND. However, an application should always call CTBFETCH in a loop until it returns CS_END_DATA.

When executing a stored procedure, the server returns any parameter values immediately after returning all row results. Therefore, an application can fetch return parameters only after processing the stored procedure row results. A stored procedure can generate several sets of row results, one for each select it contains. An application must call CTBRESULTS and CTBFETCH as many times as necessary to process these row results before calling CTBFETCH to fetch the stored procedure return parameters.

Stored procedure return status

Adaptive Server, Open Server, and Open ServerConnect applications can all return a status.

All stored procedures that run on a Adaptive Server version 4.0 or later return a status. Stored procedures usually return 0 to indicate normal completion. Open ServerConnect status values are documented under TDSNDDON and TDSTATUS in the Mainframe Connect Server Option *Programmer's Reference for PL/1*.

Because return status values are a feature of stored procedures, only an RPC or a language request containing an execute statement can generate a return status.

When executing a stored procedure, Adaptive Server returns the status immediately after returning all other results. Therefore, an application can fetch a return status only after processing the stored procedure row and parameter results,
if any.

Open Server applications return the status after any row results, but either before or after return parameters.

CTBRESULTS returns CS_STATUS_RESULT if a return status result set is available to be processed. Because a return status result set contains only a single value, one call to CTBFETCH copies the status into the program variable designated via CTBBIND. However, an application should always call CTBFETCH in a loop until it returns CS_END_DATA.

Results

Description

When a client request executes a server procedure or transaction, it can generate various types of result sets that are returned to the client application:

- Regular row results, which contain one or more rows of tabular data.
- Return parameter results, which contain a single row of return parameter data. Return parameters are values returned by stored procedures and transactions in the parameters (arguments) of the called function. For information on return parameters, see “Remote procedure calls (RPCs)” on page 44.
- Return status results, which contain a single value in a single row. For more information on a stored procedure return status, see “Remote procedure calls (RPCs)” on page 44.

Note These are the only result types supported by Open ClientConnect. Although additional result types are supported by Open Client for other platforms, they are not supported on the mainframe.

Results are returned to an application in the form of result sets. A result set contains only a single type of result data. Regular row result sets can contain multiple rows of data, but other types of result sets contain at most a single row of data.

An application processes results by calling CTBRESULTS, which indicates the type of result available by setting the *RESULT_TYP* argument. The application calls CTBRESULTS once for each result row. CTBRESULTS returns CS_CMD_DONE in *RESULT_TYP* to indicate that a result set processed completely.

Some requests, such as a language request containing a Transact-SQL update statement, do not generate results. CTBRESULTS returns CS_CMD_SUCCEED to indicate the success of a request that does not return results.

SERVERMSG structure

Description

A SERVERMSG (server message) structure contains information about an error or informational message returned by the server. This structure is defined within the application. CTBDIAG returns a message string and information about the message in this structure.

Client messages are returned to a CLIENTMSG structure, described in "CLIENTMSG structure" on page 24.

CLIENTMSG and SERVERMSG structures are part of the Mainframe ClientConnect (MCC) CTPUBLIC copybook.

This structure contains information about all messages received by the client application, including MCC messages, messages returned by the remote transactions, and messages returned by the database (such as DB2 Access Module messages and Adaptive Server messages).

Definition

A SERVERMSG structure is defined as follows:

```
DCL
   01 SERVER-MSG,
      05 SMSG_MSGNO          FIXED BIN(31),
      05 SMSG_STATE          FIXED BIN(31),
      05 SMSG_SEVERITY       FIXED BIN(31),
      05 SMSG_TEXT           CHAR(256),
      05 SMSG_TEXT_LEN       FIXED BIN(31),
      05 SMSG_SVRNAME        CHAR(256),
      05 SMSG_SVRNAME_LEN    FIXED BIN(31),
      05 SMSG_PROC           CHAR(256),
      05 SMSG_PROC_LEN       FIXED BIN(31),
      05 SMSG_LINE           FIXED BIN(31),
      05 SMSG_STATUS          FIXED BIN(31);
```

- SMSG_MSGNO is the server message number. This field corresponds to the *MESSAGE_NUMBER* argument of the Gateway-Library function TDSNDMSG.
- SMSG_STATE is the message state. This field corresponds to the *ERROR_STATE* argument of the Gateway-Library function TDSNDMSG.
- SMSG_SEVERITY is a symbolic value representing the severity of the message. Severity values are provided in the CTPUBLIC copybook. This field corresponds to the *SEVERITY* argument of the Gateway-Library function TDSNDMSG.

  Table 2-9 on page 49 lists the legal values for SMSG_SEVERITY.

Table 2-9: Values for the SERVERMSG SMSG_SEVERITY field

SMSG_SEVERITY value	Meaning
CS_SV_INFORM (0)	No error occurred. The message is informational.
CS_SV_API_FAIL (1)	A Client-Library routine generated an error. This error is typically caused by a bad parameter or calling sequence. The server connection is probably salvageable.
CS_SV_RETRY_FAIL (2)	An operation failed, but the operation can be retried.
CS_SV_RESOURCE_FAIL (3)	A resource error occurred. This error is typically caused by an allocation error, a lack of file descriptors, or timeout error. The server connection is probably not salvageable.
CS_SV_CONFIG_FAIL (4)	A configuration error occurred.
CS_SV_COMM_FAIL (5)	An unrecoverable error in the server communication channel occurred. The server connection is not salvageable.
CS_SV_INTERNAL_FAIL (6)	An internal Client-Library error occurred.
CS_SV_FATAL (7)	A serious error occurred. All server connections are unusable.

- SMSG_TEXT is the text of the message string. This field corresponds to the *MESSAGE_TEXT* argument of the Gateway-Library function TDSNDMSG.
- SMSG_TEXT_LEN is the length, in bytes, of SMSG_TEXT. If there is no message text, the value of SMSG_TEXT_LEN is 0. This field corresponds to the *MESSAGE_LENGTH* argument of the Gateway-Library function TDSNDMSG.
- SMSG_SVRNAME is the name of the server that generated the message. This is the server name from the Server Path Table.

The Server Path Table contains the information needed by Client-Library programs to route requests to a remote server, including the name of the server and connections to use to access that server. This table is part of the Connection Router, described in the Mainframe Connect Client Option *Installation and Administration Guide*.

- SMSG_SVRNAME_LEN is the length, in bytes, of SMSG_SVRNAME.
- SMSG_PROC is the name of the remote procedure or transaction that returned the message—the name of the Adaptive Server stored procedure or the transaction ID of the mainframe transaction. This field corresponds to the *TRANSACTION_ID* argument of the Gateway-Library function TDSNDMSG.
- SMSG_MSGNO is the length, in bytes, of SMSG_PROC. This field corresponds to the *TRANSACTION_ID_LENGTH* argument of the Gateway-Library function TDSNDMSG.
- SMSG_LINE is the line number in the called procedure or transaction where the error occurred. It may also be used for miscellaneous information. This field corresponds to the *LINE_ID* argument of the Gateway-Library function TDSNDMSG.
- SMSG_STATUS is reserved for future use.

SQLCA structure

Description

A SQLCA structure can be used in conjunction with CTBDIAG to retrieve Client-Library and server error and informational messages.

Definition

A SQLCA structure is defined as follows:

```
DCL
01 SQLCA_MSG,
     05 SQLCAID CHAR(8),
     05 SQLCABC FIXED BIN(31),
     05 SQLCODE FIXED BIN(31),
     05 SQLERRM,
     49 SQLERRML FIXED BIN(31),
     49 SQLERRMC CHAR(256),
     05 SQLERRP CHAR(8),
     05 SQLERRD OCCURS 6 TIMESFIXED BIN(31),
     05 SQLWARN,
     10 SQLWARN0 CHAR(1),
     10SQLWARN1 CHAR(1),
```

```
10SQLWARN2 CHAR(1),
10 SQLWARN3 CHAR(1),
10 SQLWARN4 CHAR(1),
10 SQLWARN5 CHAR(1),
10 SQLWARN6 CHAR(1),
10 SQLWARN7 CHAR(1),
05 SQLEXT   CHAR(8);
```

- *SQLCAID* is “SQLCA” (This value is automatically provided).
- *SQLCABC* is ignored.
- *SQLCODE* is the server or Client-Library message number. For information on how Client-Library maps message numbers to *SQLCODE*, see “SQLCODE structure” on page 52. For a list of gateway messages, see the Mainframe Connect Client Option *Installation and Administration Guide* shipped with this product.
- *SQLERRML* is the length of the actual message text (not the length of the text placed in SQLERRMC).
- *SQLERRMC* is the null-terminated text of the message. If the message is too long for the array, Client-Library truncates it before appending the null terminator.
- *SQLERRP* is the first eight characters of the stored procedure being executed at the time of the error.
- *SQLERRD* is the number of rows successfully inserted, updated, or deleted before the error occurred.
- *SQLWARN* is an array of warnings:
  - If *SQLWARN0* is blank, all other SQLWARN variables are blank. If *SQLWARN0* is not blank, at least one other *SQLWARN* variable is set to W.
  - If *SQLWARN1* is W, Client-Library truncated at least one column’s value when storing it into a mainframe variable.
  - If *SQLWARN2* is W, at least one null value was eliminated from the argument set of a function.
  - If *SQLWARN3* is W, the number of mainframe variables specified in the into clause of a select statement is not equal to the number of result columns.
  - If *SQLWARN4* is W, a dynamic SQL update or delete statement did not include a where clause.

- If *SQLWARN5* is W, a server conversion or truncation error occurred. *SQLEXT* is ignored.

SQLCODE structure

Description

A SQLCODE structure can be used in conjunction with CTBDIAG to retrieve Client-Library and server error and informational messages. A SQLCODE structure can be located anywhere and mapped to SQLCA.

A SQLCODE structure is defined as a 4-byte integer.

Client-Library always sets SQLCODE and the SQLCODE field of the SQLCA structure identically. (See "SQLCA structure" on page 50.)

Mapping server messages to SQLCODE

A server message number is mapped to a SQLCODE of 0 if it has a severity of 0.

Other server messages can be mapped to a SQLCODE of 0 as well.

Server message numbers are negated before being placed into SQLCODE. This ensures that SQLCODE is negative if an error occurs.

For a list of server messages returned by gateway products (Mainframe ClientConnect, Open ServerConnect, or OmniSQL Access Module for DB2), see the Mainframe Connect Client Option *Installation and Administration Guide* for any of these products.

Mapping Client-Library messages to SQLCODE

The Client-Library message "No rows affected" is mapped to a SQLCODE of 100.

Client-Library messages with CS_SV_INFORM severities are mapped to a SQLCODE of 0.

Other Client-Library messages may be mapped to a SQLCODE of 0 as well.

Client-Library message numbers are negated before being placed into SQLCODE. This ensures that SQLCODE is negative when an error occurs.

For a list of Client-Library messages, see the Mainframe Connect Client Option *Installation and Administration Guide*.

Handles

An application needs to call Open ClientConnect functions to allocate, use, and deallocate most handles, but does not need to access them directly.

Client-Library uses handles at three levels. Each handle defines and manages a particular environment. Each type of handle can have certain properties, described below.

Note Most Client-Library functions include a handle argument. An application must allocate these handles before using them as arguments.

Types of handles

The following handles are used with Client-Library:

- *Context handle*. A context handle defines a particular application, context, or operating environment. The context handle is defined in the program call CSBCTXALLOC.

  An application can have only one context.

  A context handle corresponds to the IHANDLE structure in the Open ServerConnect Gateway-Library.

  The context handle can have the following properties listed in Table 2-10.

Table 2-10: Context properties

Property	Set By
CS_EXTRA_INF	CSBCONFIG
CS_LOGIN_TIMEOUT	CTBCONFIG
CS_MAX_CONNECT	CTBCONFIG
CS_NETIO	CTBCONFIG
CS_NOINTERRUPT	CTBCONFIG
CS_TEXTLIMIT	CTBCONFIG
CS_TIMEOUT	CTBCONFIG
CS_VERSION	CSBCONFIG

- Connection handle.

  This is the handle for an individual client/server connection. The connection handle is defined in the program's CTBCONALLOC call. If parallel sessions are used, there must be one connection handle for each session. An application can have up to 25 connections.

Open ClientConnect uses a Connection Router program to define connections. Each connection handle corresponds to a connection defined with the Connection Router. For details about the Connection Router, see the Mainframe Connect Client Option *Installation and Administration Guide*.

A connection handle can have the following properties listed in Table 2-11 on page 54.

Table 2-11: Connection properties

Property	Set by
CS_APPNAME	CTBCONPROPS
CS_CHARSETCNV	CTBCONPROPS
CS_COMMBLOCK	CTBCONPROPS
CS_EXTRA_INF	CTBCONPROPS
CS_HOSTNAME	CTBCONPROPS
CS_LOC_PROP	CTBCONPROPS
CS_LOGIN_STATUS	CTBCONPROPS
CS_NET_DRIVER	CTBCONPROPS
CS_NETIO	CTBCONPROPS
CS_NOINTERRUPT	CTBCONPROPS
CS_PACKETSIZE	CTBCONPROPS
CS_PASSWORD	CTBCONPROPS
CS_TDS_VERSION	CTBCONPROPS
CS_TEXTLIMIT	CTBCONPROPS
CS_TRANSACTION_NAME	CTBCONPROPS
CS_USERNAME	CTBCONPROPS

- Command handle.

  A command handle defines a command space, which is used to send commands to a server over a connection and process the results. A command handle is defined in the program call CTBCMDALLOC. Each command handle is associated with a particular connection. There can be any number of command handles associated with a connection.

  A command handle and its associated connection handle correspond to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

  A command handle can have the CS_USERDATA property.

Routines that affect handles

Table 2-12 lists the routines that allocate, use, and deallocate handles.

Table 2-12: Routines that manipulate hidden structures

Structure	Allocated and used by
CONTEXT	CSBCTXALLOC, CTBCONFIG, CSBCONFIG, CSBCTXDROP
CONNECTION	CTBCONALLOC, CTBCONPROPS, CTBCONDROP
COMMAND	CTBCMDALLOC, CTBCMDPROPS, CTBCMDDROP

CHAPTER 3

Functions

This chapter describes the functions that are included with your Open ClientConnect software.

Table 3-1: List of Functions

Function	Description
CTBBIND (see CTBBIND on page 59)	Binds a returned column or parameter to a program variable.
CTBCANCEL (see CTBCANCEL on page 70)	Cancels a request or the results of a request.
CTBCLOSE (see CTBCLOSE on page 72)	Closes a server connection.
CTBCMDALLOC (see CTBCMDALLOC on page 75)	Allocates a command handle.
CTBCMDDROP (see CTBCMDDROP on page 78)	Deallocates a command handle.
CTBCMDPROPS (see CTBCMDPROPS on page 81)	Sets, retrieves, or clears information about the current result set.
CTBCOMMAND (see CTBCOMMAND on page 84)	Initiates a language request or remote procedure call.
CTBCONALLOC (see CTBCONALLOC on page 88)	Allocates a connection handle.
CTBCONDROP (see CTBCONDROP on page 94)	Deallocates a connection handle.
CTBCONFIG (see CTBCONFIG on page 98)	Sets or retrieves context properties.
CTBCONNECT (see CTBCONNECT on page 102)	Connects to a server.
CTBCONPROPS (see CTBCONPROPS on page 105)	Sets or retrieves connection handle properties.
CTBDESCRIBE (see CTBDESCRIBE on page 112)	Returns a description of result data.
CTBDIAG (see CTBDIAG on page 119)	Manages in-line error handling.
CTBEXIT (see CTBEXIT on page 136)	Exits the programming interface.
CTBFETCH (see CTBFETCH on page 139)	Fetches result data.
CTBGETFORMAT (see CTBGETFORMAT on page 145)	Returns the server-defined format for a result column.
CTBINIT (see CTBINIT on page 147)	Initializes the programming interface.
CTBPARAM (see CTBPARAM on page 149)	Defines a command parameter.
CTBREMOTEPWD (see CTBREMOTEPWD on page 158)	Defines or clears passwords to be used for server-to-server connections.
CTBRESINFO (see CTBRESINFO on page 161)	Returns result set information.
CTBRESULTS (see CTBRESULTS on page 167)	Sets up result data to be processed.
CTBSEND (see CTBSEND on page 174)	Sends a request to the server.
CSBCONFIG (see CSBCONFIG on page 180)	Sets or retrieves global context properties.

Function	Description
CSBCONVERT (see CSBCONVERT on page 183)	Converts a data value from one datatype to another.
CSBCTXALLOC (see CSBCTXALLOC on page 190)	Allocates a context structure.
CSBCTXDROP (see CSBCTXDROP on page 192)	Deallocates a context structure.

Note In the “Parameters” section for the following function descriptions, (I) indicates an input parameter, and (O) indicates an output parameter.

CTBBIND

Description

Associates a returned column, parameter or status with a program variable.

Syntax

```
%INCLUDE CTPUBLIC;
DCL
    01 COMMAND          FIXED BIN(31) INIT(0);
    01 RETCODE          FIXED BIN(31) INIT(0);
    01 ITEM_NUM         FIXED BIN(31) INIT(1);
    01 DATAFMT,
       05 FMT_NAME      CHAR(132),
       05 FMT_NAMELEN   FIXED BIN(31),
       05 FMT_TYPE      FIXED BIN(31),
       05 FMT_FORMAT    FIXED BIN(31),
       05 FMT_MAXLEN    FIXED BIN(31),
       05 FMT_SCALE     FIXED BIN(31),
       05 FMT_PRECIS    FIXED BIN(31),
       05 FMT_STATUS    FIXED BIN(31),
       05 FMT_COUNT     FIXED BIN(31),
       05 FMT_UTYPE     FIXED BIN(31),
       05 FMT_LOCALE    FIXED BIN(31);
    01 BUFFER type;
    01 COPIED           FIXED BIN(31);
    01 COPIED_NULL      FIXED BIN(31) INIT(0);
    01 INDICATOR        FIXED BIN(15) INIT(0);
    01 INDICATOR_NULL   FIXED BIN(31) INIT(0);

CALL CTBBIND (COMMAND, RETCODE, ITEM_NUM, DATAFMT,
BUFFER, COPIED, COPIED_NULL, INDICATOR, INDICATOR_NULL);
```

Parameters

COMMAND

(I) Handle for this connection. This is the handle defined in the CTBCMDALLOC call for this connection. The command handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

ITEM_NUM

(I) Ordinal number of the result column, return parameter, or return status value that is to be bound.

When binding a result column, *ITEM_NUM* is the column number. For example, the first column in the select list of a SQL select statement is column number 1, the second is column number 2, and so on.

When binding a return parameter, *ITEM_NUM* is the ordinal rank of the return parameter. The first parameter returned by a procedure or parameter is number 1. Adaptive Server stored procedure return parameters are returned in the order originally specified in the create procedure statement for the stored procedure. This is not necessarily the same order as specified in the RPC that invoked the stored procedure or transaction.

In determining what number to assign to *ITEM_NUM*, do not count non-return parameters. For example, if the second parameter in a stored procedure is the only return parameter, its *ITEM_NUM* is 1.

When binding a stored procedure return status, *ITEM_NUM* must be 1. There is only one column and one row in a return status result set.

To clear all bindings, assign *ITEM_NUM* a value of CS_UNUSED.

DATAFMT

(I) A structure that contains a description of the destination variable(s). This structure is also used by CTBDESCRIBE, CTBPARAM and CSBCONVERT and is explained in the section "DATAFMT structure" on page 26.

Table 3-2 lists the fields in the *DATAFMT* structure, indicates whether they are used by CTBBIND, and contains general information about each field. CTBBIND ignores *DATAFMT* fields that it does not use.

Warning! You must initialize the entire *DATAFMT* structure to zeroes. Failure to do so causes addressing exceptions.

Table 3-2 lists the fields in the *DATAFMT* structure for CTBBIND.

Table 3-2: Fields in the DATAFMT structure for CTBBIND

Field	When used	Value represents
FMT_NAME	Not used (CS_FMT_UNUSED).	Not applicable.

Field	When used	Value represents
FMT_NAMELEN	Not used (CS_FMT_UNUSED).	Not applicable.
FMT_TYPE	When binding all types of results.	The datatype of the destination variable (*BUFFER*). All datatypes listed under "Datatypes" on page 30 are valid. CTBBIND supports a wide range of datatype conversions, so FMT_TYPE can be different from the datatype returned by the server. For instance, by specifying a datatype of CS_FLOAT, you can bind a CS_MONEY or CS_MONEY4 value to a float-type program variable. The appropriate data conversion happens automatically. A return status always has a datatype of CS_INT.
FMT_FORMAT	When binding results to fixed-length character or binary destination variables. In all other cases, this field is unused (CS_FMT_UNUSED).	The destination format of character or binary data. For fixed-length character-type destinations only: CS_FMT_PADBLANK pads to the full length of the variable with blanks. For fixed-length character or binary type destination variables: CS_FMT_PADNULL pads to the full length of the variable with zeroes.
FMT_MAXLEN	When binding all types of results to non-fixed-length types. FMT_MAXLEN is ignored when binding to fixed-length datatypes.	The length of the destination variable, in bytes. If *BUFFER* has more than one element (that is, it is an array), FMT_MAXLEN is the length of one element. When binding to character or binary destinations, FMT_MAXLEN must describe the total length of the destination variable, including any space required for special terminating bytes, with this exception: when binding to a VARYCHAR-type destination such as DB2's VARCHAR, FMT_MAXLEN does not include the length of the "LL" length specification. To clear bind values, assign FMT_MAXLEN a value of 0. If the length specified in FMT_MAXLEN is too small to hold a result data item, then, at fetch time, CTBFFETCH will discard the result item that is too large, fetch any remaining items in the row, and return CS_ROW_FAIL. If this occurs, the contents of *BUFFER* will be undefined. When binding Sybase-numerical/decimal to char, use CTDESCRIBE to determine precision. FMT_MAXLEN should be precision + 2 in this case. When binding to packed decimal CTBBIND calculates FMT_MAXLEN as (precision/2) + 1.

Field	When used	Value represents
FMT_SCALE	Only when converting column results or return parameters to or from an Open ServerConnect packed decimal (CS_PACKED370), Sybase-decimal, and Sybase-numeric datatypes.	The number of digits to the right of the decimal point. If the source value is the same datatype as the destination value, set FMT_SCAL to CS_SRC_VALUE to indicate that the destination variable should pick up the value for FMT_SCALE from the source data. FMT_SCALE must be less than or equal to FMT_PRECIS and cannot be greater than 31. If the actual scale is greater than the scale specified in FMT_SCALE but not greater than 31, CTBBIND truncates the results and issues a warning. If the actual scale is greater than 31, the CTBBIND call fails. When binding sybase-numeric/decimal to char or packed-decimal use CTDESCRIBE to determine precision and scale.
FMT_PRECIS	Only when converting column results or return parameters to an Open ServerConnect packed decimal (CS_PACKED370), Sybase-decimal, and Sybase-numeric datatypes.	The total number of decimal digits in the destination variable. This is the *n* in the *BUFFER* declaration: `FIXED BIN (N)` If the source data is the same datatype as the destination variable, setting FMT_PRECIS to CS_SRC_VALUE instructs the destination variable to pick up its value for FMT_PRECIS from the source data. If the precision of the value fetched exceeds the precision of the destination variable, CTBFETCH returns a warning message. FMT_PRECIS must be greater than or equal to FMT_SCALE and cannot be less than 1 or greater than 31.
FMT_STATUS	Not used (CS_FMT_UNUSED).	Not applicable.
FMT_COUNT	When binding all types of results. Only regular row result sets ever contain multiple rows. Other types of results (for example, return parameters, status) are treated like a single row of results.	The number of result rows to be copied to program variables per CTBFETCH call. If FMT_COUNT is larger than the number of available rows, only the available rows are copied. FMT_COUNT must have the same value for all columns in a result set: – If FMT_COUNT is 0 or 1, 1 row is fetched; – If FMT_COUNT is greater than 1, it represents the number of rows that are fetched. In this case, *BUFFER* must be an array. **Note** Only regular row result sets can contain multiple rows. Other types of results (such as return parameters and status) are treated like a single row of results.
FMT_UTYPE	Not used (CS_FMT_UNUSED).	Not applicable.

Field	When used	Value represents
FMT_LOCALE	Not used (CS_FMT_UNUSED).	Reserved for future use.

BUFFER

(I) Destination variable. A single field or an array of *n* elements where *n* is FMT_COUNT. Each array element is of size FMT_MAXLEN.

BUFFER is the program variable to which CTBBIND binds the server results. When the application calls CTBFETCH to fetch the result data, it is copied into this space.

This argument is typically one of the following datatypes:

```
01 BUFFER    FIXED BIN(n);
01 BUFFER    CHAR(n);
```

If you no longer want to store incoming data in this buffer, set FMT_MAXLEN to 0. This clears the binding.

COPIED

(O) Length of the incoming data. This can be a single field or, if *BUFFER* is an array, it can be an array of *n* elements where *n* is FMT_COUNT. At fetch time, CTBFETCH fills *COPIED* with the length(s) of the copied data.

COPIED_NULL

(I) NULL indicator for *COPIED*. This argument allows you to indicate that *COPIED* should be treated as null (zeroes). Assign this argument one of the following values:

Value	Meaning
CS_PARAM_NULL (-102)	*COPIED* is zeroes. If *COPIED* is an array, assigning CS_PARAM_NULL to this argument causes all elements of *COPIED* to be treated as zeroes.
CS_PARAM_NOTNULL (-103)	*COPIED* is not zeroes.

INDICATOR

(O) From 1 to the value of FMT_COUNT integer variables. At fetch time, CTBFETCH uses each variable to indicate the following conditions about the fetched data:

Value	Integer value	Meaning
CS_NULLDATA	-1	There was no data to fetch. In this case, no data is copied to the destination variable.
CS_GOODDATA	0	The fetch was successful.

If *BUFFER* is an array, *INDICATOR* will also be an array.

INDICATOR_NULL

(I) NULL indicator for *INDICATOR*. This argument allows you to treat *INDICATOR* as null (zeroes). Assign this argument one of the following values:

Value	Meaning
CS_PARAM_NULL (-102)	INDICATOR is zeroes. If INDICATOR is an array, assigning CS_PARAM_NULL to this argument causes all elements of INDICATOR to be treated as zeroes.
CS_PARAM_NOTNULL (-103)	INDICATOR is not zeroes.

Return value

CTBBIND returns one of the following values listed in Table 3-3.

Table 3-3: CTBBIND return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_CONNECTION_TERMINATED (-4997)	The connection is not active.
TDS_INVALID_DATAFMT_VALUE (-181)	DATAFMT field contains an illegal value.
TDS_INVALID_PARAMETER (-4)	A parameter was given an illegal value.
TDS_INVALID_VAR_ADDRESS (-175)	This value cannot be NULL.
TDS_NO_COMPUTES_ALLOWED (-60)	Compute results are not supported.
TDS_RESULTS_CANCELED (-49)	A cancel was sent to purge results.
TDS_SOS (-257)	Memory shortage. The operation failed.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call.

Examples

This code fragment demonstrates the use of CTBBIND to bind returned data to program variables. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to bind each data                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/
 BIND_COLUMNS: PROC ;
         CALL CTBDESCR( CSL_CMD_HANDLE,
                        CSL_RC,
                        PARM_CNT,
                        DATAFMT ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBDESCRIBE failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE;
         END ;

/*-------------------------------------------------------------*/
/* We need TO bind the data TO program variables. We don't     */
/* care about the indicator variable so we'll pass NULL for    */
/* that PARAMeter in OC_BIND().                                */
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* rows per fetch                                              */
/*-------------------------------------------------------------*/

         DF_COUNT = 1 ;

         SELECT( DF_DATATYPE ) ;

 /*-------------------------------------------------------------*/
 /* bind the first column, FIRSTNME defined as VARCHAR(12)      */
 /*-------------------------------------------------------------*/
           WHEN( CS_VARCHAR_TYPE )
           DO ;
             DF_DATATYPE   = CS_VARCHAR_TYPE;
             DF_FORMAT     = CS_FMT_UNUSED;
             DF_MAXLENGTH  = STG(CF_COL_FIRSTNME) - 2;
             DF_COUNT      = 1;
             CF_COL_NUMBER = 1;
```

```
            CALL CTBBIND( CSL_CMD_HANDLE,
                          CSL_RC,
                          CF_COL_NUMBER,
                          DATAFMT,
                          CF_COL_FIRSTNME,
                          CF_COL_LEN,
                          CS_PARAM_NOTNULL,
                          CF_COL_INDICATOR,
                          CS_PARAM_NULL);

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBBIND CS_VARCHAR_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
            END ;
          END ;

/*-------------------------------------------------------------*/
/* bind the second column, EDLEVEL defined as SMALLINT         */
/*-------------------------------------------------------------*/
          WHEN( CS_SMALLINT_TYPE )
          DO ;
            DF_DATATYPE   = CS_SMALLINT_TYPE;
            DF_FORMAT     = CS_FMT_UNUSED;
            DF_MAXLENGTH  = STG(CF_COL_EDLEVEL);
            DF_COUNT      = 1;
            CF_COL_NUMBER = 2;

            CALL CTBBIND( CSL_CMD_HANDLE,
                          CSL_RC,
                          CF_COL_NUMBER,
                          DATAFMT,
                          CF_COL_EDLEVEL,
                          CF_COL_LEN,
                          CS_PARAM_NOTNULL,
                          CF_COL_INDICATOR,
                          CS_PARAM_NULL ) ;

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBBIND CS_SMALLINT_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
```

```
            END ;
          END ;

          OTHERWISE ;

        END ; /* end of SELECT( DF_DATATYPE ) */

END BIND_COLUMNS ;
```

Usage

- CTBBIND associates (“binds”) a column, parameter, or status returned by a server to a program variable. Once a result is bound to a variable, any information returned in that column or parameter, or any status returned during a CTBFETCH call is copied to that variable.
- An application must call CTBBIND once for each result column or return parameter.
- CTBBIND can be used to bind a result column, a return parameter, or a stored procedure status value. When binding a result column, a single call to CTBBIND can bind multiple rows of the column. When binding a return status, you must bind a single variable of type integer.
- An application calls CTBBIND after CTBRESULTS and before CTBFETCH. CTBRESULTS tells the application whether there are any results to be bound and if so, what kind; CTBFETCH retrieves the results and copies them into the bound variable.
- CTBBIND binds only the current result type. CTBRESULTS indicates the current result type via its *RESULT_TYP* argument. For example, if CTBRESULTS returns CS_STATUS_RESULT, a return status is available for binding.
- An application can call CTBRESINFO to determine the number of items in the current result set, and can call CTBDESCRIBE to get a description of each item.
- An application can only bind a result item to a single program variable. If an application binds a result item to multiple variables, only the last binding takes effect.
- Binding for a particular type of result remains in effect until CTBRESULTS returns CS_CMD_DONE to indicate that the results of a logical command are processed completely.
- If you no longer want to store incoming data in the program variable, call CTBBIND with a zero-length *BUFFER* (for example, FMT_MAXLEN = 0).

- An application can rebind while actively fetching rows. That is, an application can call CTBBIND inside a CTBFETCH loop if it needs to change the binding of a result item (This action is not recommended).
- Table 3-4 on page 68 lists the conversions performed by CTBBIND.

Table 3-4: Datatype conversions performed by CTBBIND

Source type	Result type
CS_VARCHAR	CS_CHAR
CS_CHAR	CS_VARCHAR
CS_MONEY	CS_CHAR
CS_MONEY	CS_VARCHAR
CS_FLT4	CS_FLT8
CS_MONEY	CS_FLT8
CS_PACKED370	CS_FLT8
CS_FLT8	CS_FLT4
CS_CHAR	CS_PACKED370
CS_VARCHAR	CS_PACKED370
CS_MONEY	CS_PACKED370
CS_FLT8	CS_PACKED370
CS_NUMERIC	CS_CHAR
CS_DECIMAL	CS_CHAR
CS_PACKED370	CS_DECIMAL
CS_NUMERIC	CS_PACKED370
CS_DECIMAL	CS_PACKED370
CS_DATETIME	CS_CHAR

Array binding

- Array binding is the act of binding a result column to an array of program variables. At fetch time, multiple rows of the column are copied to the array of variables with a single CTBFETCH call. An application indicates array binding by assigning FMT_COUNT a value greater than 1.
- Array binding is only practical for regular row results. Other types of results are considered to be the equivalent of a single row.
- When binding columns to arrays in a single command, all CTBBIND calls in the sequence of calls binding the columns must use the same value for FMT_COUNT. For example, when binding three columns to arrays, it is an error to assign FMT_COUNT a value of 5 in your first two CTBBIND calls and a value of 3 in the last.

- CTBBIND supports CS_NUMERIC and CS_DECIMAL datatypes.
- Use CTDESCRIBE before CTBBIND with decimal datatypes to get correct precision and scale.

See also

Related functions:

- CTBCANCEL on page 70
- CTBCOMMAND on page 84

Related topics:

- “Datatypes” on page 30

CTBCANCEL

Description

Cancels a request or the results of a request.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION      FIXED BIN(31) INIT(0);
   01 RETCODE         FIXED BIN(31) INIT(0);
   01 COMMAND         FIXED BIN(31) INIT(0);
   01 CANCELTYPE      FIXED BIN(31);

CALL CTBCANCE (CONNECTION, RETCODE, COMMAND,
CANCELTYPE);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

Either *CONNECTION* or *COMMAND* must be null (zeroes). If *CONNECTION* is supplied and *COMMAND* is empty, the cancel operation applies to all commands pending for this connection.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call. The command handle also corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

Either *CONNECTION* or *COMMAND* must be zeroes. If *COMMAND* is supplied and *CONNECTION* is zeroes, the cancel operation applies only to the command pending for this command structure.

CANCELTYPE

(I) Type of cancel requested. The following table lists the symbolic values that are legal for *CANCELTYPE*:

Value	Meaning
CS_CANCEL_ALL (6001) or CS_CANCEL_ATTN (6002)	CTBCANCEL sends an attention to the server, instructing it to cancel the current request, and immediately discards all results generated by the request.

Return value

CTBCANCEL returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_CONNECTION_TERMINATED (-4997)	The connection is not active.
TDS_INVALID_TDPROC (-18)	Specified command handle is invalid.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call.

Usage

- CTBCANCEL cancels the current result set.
- Canceling the current result set is equivalent to discarding the current set of results. Once results are discarded, they are no longer available to an application.
- In Open ClientConnect, CS_CANCEL_ALL and CS_CANCEL_ATTN function identically. Both immediately cancel the current request and discard all results generated by it.

Canceling a request

- To cancel the current request and all results generated by it, an application calls CTBCANCEL with *CANCELTYPE* as CS_CANCEL_ATTN or CS_CANCEL_ALL. These calls tell Client-Library to:
  - Discard all results already generated by the request.
  - Send an attention to the server instructing it to halt execution of the current request.

  For example, suppose the current request is a Transact-SQL language request that contains the queries:

  ```
  select * from titles
  select * from authors
  ```

- If an application cancels the language request after the first query executes but before the second query executes:
  - All remaining results from the first query are discarded.

- Execution of the second query is halted.

Note A call to CTBCANCEL with *CANCELTYPE* as CS_CANCEL_ALL or CS_CANCEL_ATTN must be immediately followed by a CTBRESULTS call.

- In Open Client Client-Library, canceling with *CANCELTYPE* as CS_CANCEL_ALL or CS_CANCEL_ATTN leaves the command structure in a "clean" state, available to be used for another operation.
- For both the CS_CANCEL_ATTN and CS_CANCEL_ALL types of cancels, if no request is in progress, CTBCANCEL returns CS_SUCCEED immediately.
- If a request initiates but has not been sent, a CS_CANCEL_ALL is rejected.

See also

Related functions:

- CTBFETCH on page 139
- CTBRESULTS on page 167

CTBCLOSE

Description

Closes a server connection.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION    FIXED BIN(31) INIT(0);
   01 RETCODE       FIXED BIN(31) INIT(0);
   01 OPTION        FIXED BIN(31);

 CALL CTBCLOSE (CONNECTION, RETCODE, OPTION);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

OPTION

(I) Option, if any, to use for the close. The following table lists the symbolic values that are legal for *OPTION*:

Value	Meaning
CS_UNUSED (-99999)	CTBCLOSE logs out and closes the connection. If the connection has results pending, CTBCLOSE returns CS_FAIL. This is the default behavior.
CS_FORCE_CLOSE (302)	CTBCLOSE closes the connection whether or not results are pending, and without notifying the server. This option is primarily for use when an application hangs waiting for a server response.
CS_KEEP_CON	This option is ignored. CICS treats it like CS_UNUSED.

Return value

CTBCLOSE returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. The most common reason for a CTBCLOSE failure is pending results on the connection.
TDS_CONNECTION_TERMINATED (-4997)	The connection is not active.
TDS_COMMAND_ACTIVE (-7)	A command is in progress.
TDS_RESULTS_STILL_ACTIVE (-50)	Some results are still pending.

Examples

The following code fragment demonstrates how CTBCLOSE is used with other functions at the end of a program to close the connection and return to CICS. It is taken from the sample program SYCTSAA4 in Appendix A, “Sample Language Application.”

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to perform drop command handler, close server          */
/* connection, and deallocate Connection Handler.                    */
/*                                                                   */
/*-------------------------------------------------------------------*/
CLOSE_CONNECTION: PROC ;

/*-------------------------------------------------------------------*/
/* drop the command handle                                           */
/*-------------------------------------------------------------------*/
```

```
        CALL CTBCMDDR( CSL_CMD_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCMDDROP failed' ;
          CALL ERROR_OUT ;
        END ;

/*-------------------------------------------------------------------*/
/* close the server connection                                       */
/*-------------------------------------------------------------------*/

        CALL CTBCLOSE( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCLOSE failed' ;
          CALL ERROR_OUT ;
        END ;

/*-------------------------------------------------------------------*/
/* DE_ALLOCATE THE CONNECTION HANDLE                                 */
/*-------------------------------------------------------------------*/

        CALL CTBCONDR( CSL_CON_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCONDROP failed' ;
          CALL ERROR_OUT ;
        END ;

END CLOSE_CONNECTION ;
```

Usage

- CTBCLOSE closes a server connection. All command handles associated with the connection are deallocated.
- To deallocate a connection handle, an application can call CTBCONDROP after the connection successfully closes.

- The behavior of CTBCLOSE depends on the value of *OPTION*, which determines the type of close. The following sections contain information on a type of close.

Default close behavior (OPTION is CS_UNUSED):

If the connection has any pending results, CTBCLOSE returns CS_FAIL. To correct the failure, use CTBCLOSE with the CS_FORCE_CLOSE option or read in all of your results.

Before terminating the connection with the server, CTBCLOSE sends a logout message to the server and reads the response to this message. The contents of this message do not affect the behavior of CTBCLOSE.

Forced close behavior (OPTION is CS_FORCE_CLOSE):

The connection is closed whether or not it has pending results.

Because this option sends no logout message to the server, the server cannot tell whether the close is intentional or whether it is the result of a lost connection or crashed client.

See also

Related functions:

- CTBCONDROP on page 94
- CTBCONNECT on page 102
- CTBCONPROPS on page 105

CTBCMDALLOC

Description

Allocates a command handle.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION      FIXED BIN(31) INIT(0);
   01 RETCODE         FIXED BIN(31);
   01 COMMAND         FIXED BIN(31) INIT(0);

 CALL CTBCMDAL (CONNECTION, RETCODE, COMMAND);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

COMMAND

(O) Variable where this newly-allocated command handle is returned. All subsequent client requests using this connection must use this same name in the *COMMAND* argument. The command handle also corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

In case of error, CTBCMDALLOC returns zeroes to this argument.

Return value

CTBCMDALLOC returns one of the following values listed in Table 3-5.

Table 3-5: CTBCMDALLOC return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. The most common reason for a CTBCMDALLOC failure is a lack of adequate memory.
TDS_SOS (-257)	Memory shortage. The mainframe subsystem was unable to allocate enough memory for the control block that CTBCMDALLOC was trying to create. The operation failed.

Examples

The following code fragment demonstrates how CTBCMDALLOC is used during program initialization. It is taken from the sample program SYCTSAA4 in Appendix A, “Sample Language Application.”

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to allocate, send, and process commands                */
/*                                                                   */
/*-------------------------------------------------------------------*/
  SEND_COMMAND: PROC ;
/*-----------------------------------------------------------*/
/* find out what the maximum number of connections is        */
/*-----------------------------------------------------------*/

        CALL CTBCONFI( CSL_CTX_HANDLE,
                       CSL_RC,
                       CS_GET,
                       CS_MAX_CONNECT,
                       CF_MAXCONNECT,
                       STG(CF_MAXCONNECT),
                       CS_FALSE,
                       CF_OUTLEN ) ;
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCONFIG failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;
   /*-----------------------------------------------------------*/
   /* display number of connections                             */
   /*-----------------------------------------------------------*/

        OR2_MAXCONNECT = CF_MAXCONNECT;

/*-------------------------------------------------------------*/
/* allocate a command handle                                   */
/*-------------------------------------------------------------*/

        CALL CTBCMDAL( CSL_CON_HANDLE,
                       CSL_RC,
                       CSL_CMD_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCMDALLOC failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* prepare the language request                                */
/*-------------------------------------------------------------*/

        PF_STRLEN = STG(CF_LANG2 ) ;

        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_LANG_CMD,
                       CF_LANG2,
                       PF_STRLEN,
                       CS_UNUSED ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCOMMAND failed' ;
          NO_ERRORS_SW = FALSE ;
```

```
  CALL ERROR_OUT;
  CALL ALL_DONE ;
END ;
```

Usage

- CTBCMDALLOC allocates a command handle on a specified connection. A command handle is a control structure that a Client-Library application uses to send requests to a server and process the results. Together, command and connection handles perform the functions of the Open ServerConnect *TDPROC* structure.
- Before calling CTBCMDALLOC, an application must allocate a connection structure via the Client-Library routine CTBCONALLOC.
- An application must call CTBCMDALLOC once for each logical command it issues. Each SQL statement is considered a separate logical command. For batched SQL, call CTBCMDALLOC once for each batch.

See also

Related functions:

- CTBCMDDROP on page 78
- CTBCMDPROPS on page 81
- CTBCOMMAND on page 84
- CTBCONALLOC on page 88

CTBCMDDROP

Description

Deallocates a command handle.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);

 CALL CTBCMDDR (COMMAND, RETCODE);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

Return value

CTBCMDDROP returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. CTBCMDDROP returns CS_FAIL if the command handle has any results pending.
TDS_COMMAND_ACTIVE (-7)	A command is in progress.
TDS_RESULTS_STILL_ACTIVE (-50)	Some results are still pending.

Examples

The following code fragment demonstrates how CTBCMDDROP is used with other routines at the end of a program after results have been processed. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to perform drop command handler, close server           */
/* connection, and deallocate Connection Handler.                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
CLOSE_CONNECTION: PROC ;

/*--------------------------------------------------------------------*/
/* drop the command handle                                            */
/*--------------------------------------------------------------------*/

        CALL CTBCMDDR( CSL_CMD_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCMDDROP failed' ;
          CALL ERROR_OUT ;
        END ;

/*--------------------------------------------------------------------*/
/* close the server connection                                        */
/*--------------------------------------------------------------------*/

        CALL CTBCLOSE( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
```

```
            MSGSTR = 'CTBCLOSE failed' ;
            CALL ERROR_OUT ;
          END ;

    /*-------------------------------------------------------------------*/
    /* DE_ALLOCATE THE CONNECTION HANDLE                                 */
    /*-------------------------------------------------------------------*/

          CALL CTBCONDR( CSL_CON_HANDLE,
                         CSL_RC ) ;

          IF CSL_RC = CS_FAIL THEN
          DO ;
            MSGSTR = 'CTBCONDROP failed' ;
            CALL ERROR_OUT ;
          END ;

    END CLOSE_CONNECTION ;
```

Usage

- CTBCMDDROP deallocates a command handle.
- If CTBCMDDROP is called while a command is pending (results have not all been returned), it fails. Before deallocating a command structure, an application should process or cancel any pending results.
- Once a command handle is deallocated, it cannot be reused. To allocate a new command handle, an application calls CTBCMDALLOC.

See also

Related functions:

- CTBCMDALLOC on page 75
- CTBCOMMAND on page 84

CTBCMDPROPS

Description

Sets, retrieves, or clears information about the current result set.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 COMMAND          FIXED BIN(31) INIT(0);
    01 RETCODE          FIXED BIN(31) INIT(0);
    01 ACTION           FIXED BIN(31);
    01 PROPERTY         FIXED BIN(31);
    01 BUFFER type;
    01 BUFFER_LEN       FIXED BIN(31);
    01 BUFBLANKSTRIP    FIXED BIN(31);
    01 OUTLEN           FIXED BIN(31);

CALL CTBCMDPR (COMMAND, RETCODE, ACTION, PROPERTY,
BUFFER, BUFFER_LEN, BUFBLANKSTRIP, OUTLEN);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call. The command handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

ACTION

(I) Action to be taken by this call. *ACTION* is an integer variable that indicates the purpose of this call.

Assign *ACTION* one of the following symbolic values:

Value	Meaning
CS_GET (33)	Retrieves the value of the property.
CS_SET (34)	Sets the value of the property.
CS_CLEAR (35)	Clears the value of the property by resetting the property to its Client-Library default value.

PROPERTY

(I) Symbolic name of the property for which the value is being set or retrieved. Client-Library properties are listed under “Properties” on page 37, with descriptions, possible values, and defaults.

BUFFER

(I/O) Variable (buffer) that contains the specified property value.

If *ACTION* is CS_SET, the buffer contains the value used by CTBCMDPROPS.

If *ACTION* is CS_GET, CTBCMDPROPS returns the requested information to this buffer.

If *ACTION* is CS_CLEAR, the buffer is reset to the default property value.

This argument is typically one of the following datatypes:

```
01 BUFFER     FIXED BIN(n);
01 BUFFER     CHAR(n);
```

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If *ACTION* is CS_SET and the value in the buffer is a fixed-length or symbolic value, *BUFFER_LEN* should have a value of CS_UNUSED. To indicate that the terminating character is the last non-blank character, set *BUFBLANKSTRIP* to CS_TRUE.

If *ACTION* is CS_GET and *BUFFER* is too small to hold the requested information, CTBCMDPROPS sets *OUTLEN* to the length of the requested information and returns CS_FAIL. To retrieve all the requested information, change the value of *BUFFER_LEN* to the length returned in *OUTLEN* and rerun the application.

If *ACTION* is CS_CLEAR, set this value to CS_UNUSED.

BUFBLANKSTRIP

(I) Blank stripping indicator. Indicates whether or not trailing blanks are stripped.

Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

If a property value is being set and the terminating character is the last non-blank character, assign CS_TRUE to *BUFBLANKSTRIP*.

OUTLEN

(O) Length, in bytes, of the retrieved information. *OUTLEN* is an integer variable where CTBCMDPROPS returns the length of the property value being retrieved.

When the retrieved information is larger than *BUFFER_LEN* bytes, an application uses the value of *OUTLEN* to determine how many bytes are needed to hold the information.

OUTLEN is used only when *ACTION* is CS_GET. When *ACTION* is CS_CLEAR or CS_SET, this value is zeroes.

Return value

CTBCMDPROPS returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_INVALID_PARAMETER (-4)	One or more arguments were given illegal values.
TDS_CANNOT_SET_VALUE (-43)	This property cannot be set by the application.

Usage

- CTBCMDPROPS sets or retrieves the values of properties of command handle structures.
- Command handle properties affect the behavior of an application at the command structure level.
- Some command handle properties default to the value of the property in the parent context. To find out which ones, see "Properties" on page 37.

See also

Related functions:

- CTBCMDALLOC on page 75
- CTBCONFIG on page 98
- CTBCONPROPS on page 105
- CTBRESINFO on page 161

Related topics:

- "Buffers" on page 21
- "Properties" on page 37

CTBCOMMAND

Description

Initiates a language request or remote procedure call (RPC).

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);
   01 REQTYPE      FIXED BIN(31) INIT(0);
   01 BUFFER type;
   01 BUFFER_LEN   FIXED BIN(31);
   01 OPTION       FIXED BIN(31);

CALL CTBCOMMA (COMMAND, RETCODE, REQTYPE, BUFFER,
BUFFER_LEN, OPTION);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call. The command handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

REQTYPE

(I) Type of request to initiate. The following symbolic values are legal for *REQTYPE*:

When REQTYPE is	CTBCOMMAND initiates	BUFFER contains
CS_LANG_CMD (148)	A language request.	The text of the language request.
CS_RPC_CMD (149)	A remote procedure call.	The name of the remote procedure.

BUFFER

(I) Variable (buffer) that contains the language request or RPC name.

This argument is typically one of the following datatypes:

```
01 BUFFER  FIXED BIN(n);
01 BUFFER  CHAR(n);
```

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If the value in the buffer is a fixed-length or symbolic value, assign *BUFFER_LEN* a value of CS_UNUSED.

OPTION

Option associated with this request, if any.

Currently, only RPCs take options. For language requests, assign *OPTION* a value of CS_UNUSED.

The following symbolic values are legal for *OPTION* when *REQTYPE* is CS_RPC_CMD:

Value	Meaning
CS_RECOMPILE (188)	Recompile the stored procedure before executing.
CS_NORECOMPILE (189)	Do not recompile the stored procedure before executing.
CS_UNUSED (-99999)	No options are assigned.

Return value

CTBCOMMAND returns one of the following values listed in Table 3-6.

Table 3-6: CTBCOMMAND return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_CONNECTION_TERMINATED (-4997)	The connection is not active.
TDS_INVALID_PARAMETER (-4)	A parameter contains an illegal value.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call.

Examples

The following code fragment demonstrates the use of CTBCOMMAND. It is taken from the sample program SYCTSAA4 in Appendix A, “Sample Language Application.”

```
/*-------------------------------------------------------------*/
/* allocate a command handle                                   */
/*-------------------------------------------------------------*/

          CALL CTBCMDAL( CSL_CON_HANDLE,
                         CSL_RC,
                         CSL_CMD_HANDLE ) ;

          IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR       = 'CTBCMDALLOC failed' ;
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE ;
          END ;
```

```
/*-------------------------------------------------------------*/
/* prepare the language request                                */
/*-------------------------------------------------------------*/

         PF_STRLEN = STG(CF_LANG2 ) ;

         CALL CTBCOMMA( CSL_CMD_HANDLE,
                        CSL_RC,
                        CS_LANG_CMD,
                        CF_LANG2,
                        PF_STRLEN,
                        CS_UNUSED ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBCOMMAND failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

 /*-------------------------------------------------------------*/
 /* send the language request                                   */
 /*-------------------------------------------------------------*/

         CALL CTBSEND( CSL_CMD_HANDLE,
                       CSL_RC ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBSEND failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

END SEND_COMMAND ;
```

Usage

- CTBCOMMAND initiates a language request or RPC. Initiating a request is the first step in sending it to a server.
- Sending a request to a server is a three-step process. To send a request to a server, an application must:

- Call CTBCOMMAND to initiate the request. CTBCOMMAND sets up internal structures that are used in developing a request stream to send to the server.
- Call CTBPARAM to pass parameters for the request. An application must call CTBPARAM once for each parameter in the request.
- Call CTBSEND to send the request to the server.

Language requests

- Language requests contain character strings that represent requests in a server's own language. For example, language requests to Adaptive Server can include any legal Transact-SQL command.
- A language request can be in any language, as long as the server to which it is directed can understand it. For example, Adaptive Server understands Transact-SQL, but requests to DB2 must use the DB2 version of SQL.
- If the language request string contains variables, an application can pass values for these variables by calling CTBPARAM once for each variable that the language string contains. A language request can have up to 255 parameters.
- Transact-SQL request variables must begin with a colon (:).

RPCs

- An RPC instructs a server to execute a stored procedure or transaction on either itself or a remote server.
- If an application uses an RPC to execute a stored procedure or transaction that requires parameters, the application calls CTBPARAM once for each parameter the stored procedure or transaction requires.
- After sending an RPC with CTBSEND, an application can process the stored procedure or transaction results with CTBRESULTS and CTBFETCH. The functions CTBRESULTS and CTBFETCH are used to process both the result rows generated by the stored procedure or transaction and the return parameters and status, if any.

See also

Related functions:

- CTBCMDALLOC on page 75
- CTBPARAM on page 149
- CTBSEND on page 174

Related topics:

- “Remote procedure calls (RPCs)” on page 44

CTBCONALLOC

Description

Allocates a connection handle.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONTEXT      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);
   01 CONNECTION   FIXED BIN(31) INIT(0);

 CALL CTBCONAL (CONTEXT, RETCODE, CONNECTION);
```

Parameters

CONTEXT

(I) A context structure. The context structure is defined in the program call CSBCTXALLOC. The context structure corresponds to the IHANDLE in the Open ServerConnect Gateway-Library.

If this value is invalid or nonexistent, CTBCONALLOC fails.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

CONNECTION

(O) Handle for this connection. All subsequent Client-Library calls using this connection must use this same name in their *CONNECTION* argument. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

This is the same value used to define the connection to the Open ClientConnect Connection Table.

In case of error, CTBCONALLOC returns zeroes to this argument.

Return value

CTBCONALLOC returns one of the following values listed in Table 3-7 on page 88.

Table 3-7: CTBCONALLOC return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.

Value	Meaning
CS_FAIL (-2)	The routine failed. The most common reason for a CTBCONALLOC failure is a lack of adequate memory.
TDS_SOS (-257)	Memory shortage. The mainframe subsystem was unable to allocate enough memory for the control block that CTBCONALLOC was trying to create. The operation failed.
TDS_GWLIB_NO_STORAGE (-17)	Could not get DSA for Gateway-Library.

Examples

The following code fragment demonstrates the use of CTBCONALLOC. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process input data                                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
 PROCESS_INPUT: PROC ;

/*-----------------------------------------------------------*/
/* allocate a connection to the server                       */
/*-----------------------------------------------------------*/

          CSL_CON_HANDLE = 0 ;

          CALL CTBCONAL( CSL_CTX_HANDLE,
                         CSL_RC,
                         CSL_CON_HANDLE ) ;

          IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR       = 'CTBCONALLOC failed' ;
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE ;
          END ;
     /*-----------------------------------------------------------*/
     /* alter properties of the connection for user-id            */
     /*-----------------------------------------------------------*/
          CALL CTBCONPR( CSL_CON_HANDLE,
                         CSL_RC,
                         CS_SET,
                         CS_USERNAME,
```

```
                        PF_USER,
                        PF_USER_SIZE,
                        CS_FALSE,
                        OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for user-id failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for password             */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_SET,
                        CS_PASSWORD,
                        PF_PWD,
                        PF_PWD_SIZE,
                        CS_FALSE,
                        OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for password failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for transaction          */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_SET,
                        CS_TRANSACTION_NAME,
                        PF_TRAN,
                        PF_TRANL,
                        CS_FALSE,
                        OUTLEN ) ;
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for transaction failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for Network driver       */
/*-------------------------------------------------------------*/

        SELECT;
          WHEN (PF_NETDRV = '         ')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'LU62' | PF_NETDRV = 'lu62')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'IBMTCPIP' | PF_NETDRV = 'ibmtcpip')
                NETDRIVER = CS_TCPIP ;
          WHEN (PF_NETDRV = 'INTERLIN' | PF_NETDRV = 'interlin')
                NETDRIVER = CS_INTERLINK ;
          WHEN (PF_NETDRV = 'CPIC' | PF_NETDRV = 'cpic')
                NETDRIVER = CS_NCPIC ;
          OTHERWISE
            DO;
                MSGSTR = 'Invalid Network driver entered';
                NO_ERRORS_SW = FALSE ;
                CALL ERROR_OUT;
                CALL ALL_DONE ;
            END;
        END;

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_NET_DRIVER,
                       NETDRIVER,
                       CS_UNUSED,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for Network driver failed' ;
          NO_ERRORS_SW = FALSE ;
```

```
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* setup retrieval of All Messages                             */
/*-------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_INIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      CS_UNUSED ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_INIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* set the upper limit of number of messages                   */
/*-------------------------------------------------------------*/

        PF_MSGLIMIT = 5 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_MSGLIMIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      PF_MSGLIMIT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_MSGLIMIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;
```

```
/*-------------------------------------------------------------*/
/* open connection to the server or CICS region                */
/*-------------------------------------------------------------*/

        CALL CTBCONNE( CSL_CON_HANDLE,
                       CSL_RC,
                       PF_SERVER,
                       PF_SERVER_SIZE,
                       CS_FALSE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONNECT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* invokes SEND_COMMAND routine                                */
/*-------------------------------------------------------------*/
        IF NO_ERRORS_SW
          THEN
            CALL SEND_COMMAND ;
```

Usage

- CTBCONALLOC allocates a connection handle to a Mainframe ClientConnect or another processing region (three-tier processing), or an Adaptive Server if using two-tier (gateway-less) processing.
- Before calling CTBCONALLOC, an application must:
  - Call CSBCTXALLOC to allocate a context structure.
  - Call CTBINIT to initialize Client-Library.
- Connecting to a server is a three-step process. To connect to a server, an application:
  - Calls CTBCONALLOC to obtain a connection handle.
  - Calls CTBCONPROPS to set the values of connection-specific properties, if desired.
  - Calls CTBCONNECT to create the connection and log in to the server.
- All connections created within a context pick up default property values from the parent context. An application can override these default values by calling CTBCONPROPS to set property values at the connection level.

- An application can have multiple connections to one or more servers at the same time.

  For example, an application can simultaneously have two connections to the server “mars,” one connection to the server “venus,” and one connection to a separate transaction processing region named CICX3. The context property CS_MAX_CONNECT, set by CTBCONFIG, determines the maximum number of connections allowed per context.

  Each server connection requires a separate connection handle.

- In order to send requests to a server, one or more command handles must be allocated for a connection. CTBCMDALLOC allocates a command handle.

See also

Related functions:

- CSBCTXALLOC on page 190
- CTBCLOSE on page 72
- CTBCMDALLOC on page 75
- CTBCONNECT on page 102
- CTBCONPROPS on page 105

CTBCONDROP

Description

Deallocates a connection handle.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION      FIXED BIN(31) INIT(0);
   01 RETCODE         FIXED BIN(31) INIT(0);

CALL CTBCONDR (CONNECTION, RETCODE);
```

Parameters

CONNECTION

(I) Handle for this connection. This must be the same value specified in the CTBCONALLOC call that initialized this connection.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

Return value

CTBCONDROP returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. The most common reason for a CTBCONDROP failure is that the connection is still open.
TDS_CONNECTION_TERMINATED(-4997)	The connection is not active.

Examples

The following code fragment demonstrates how CTBCONDROP is used with other functions at the end of a program to close the connection and return to CICS. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to perform drop command handler, close server           */
/* connection, and deallocate Connection Handler.                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
 CLOSE_CONNECTION: PROC ;

/*--------------------------------------------------------------------*/
/* drop the command handle                                            */
/*--------------------------------------------------------------------*/

         CALL CTBCMDDR( CSL_CMD_HANDLE,
                        CSL_RC ) ;

         IF CSL_RC = CS_FAIL THEN
         DO ;
           MSGSTR = 'CTBCMDDROP failed' ;
           CALL ERROR_OUT ;
         END ;

 /*--------------------------------------------------------------------*/
 /* close the server connection                                        */
 /*--------------------------------------------------------------------*/

         CALL CTBCLOSE( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_UNUSED ) ;

         IF CSL_RC = CS_FAIL THEN
         DO ;
           MSGSTR = 'CTBCLOSE failed' ;
```

```
          CALL ERROR_OUT ;
        END ;

 /*-------------------------------------------------------------------*/
 /* DE_ALLOCATE THE CONNECTION HANDLE                                 */
 /*-------------------------------------------------------------------*/

        CALL CTBCONDR( CSL_CON_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCONDROP failed' ;
          CALL ERROR_OUT ;
        END ;

 END CLOSE_CONNECTION ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to perform exit client library and deallocate context */
/* structure.                                                        */
/*                                                                   */
/*-------------------------------------------------------------------*/
 QUIT_CLIENT_LIBRARY: PROC ;

/*-------------------------------------------------------------------*/
/* exit the Client Library                                           */
/*-------------------------------------------------------------------*/

        CALL CTBEXIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_UNUSED ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBEXIT failed' ;
          CALL ERROR_OUT ;
        END ;
 /*-------------------------------------------------------------------*/
 /* de-allocate the context stricture                                 */
 /*-------------------------------------------------------------------*/

        CALL CSBCTXDR( CSL_CTX_HANDLE,
                       CSL_RC ) ;
```

```
            IF CSL_RC = CS_FAIL THEN
            DO ;
              MSGSTR = 'CSBCTXDROP failed' ;
              CALL ERROR_OUT ;
            END ;

            EXEC CICS RETURN ;

    END QUIT_CLIENT_LIBRARY ;
```

Usage

- CTBCONDROP deallocates a connection handle and all command handles associated with that connection.
- Once a connection handle is deallocated, it cannot be reused. To allocate a new connection handle, an application calls CTBCONALLOC.
- An application cannot deallocate a connection handle until the connection it represents successfully closes. To close a connection, an application calls CTBCLOSE.

See also

Related functions:

- CTBCLOSE on page 72
- CTBCONALLOC on page 88
- CTBCONNECT on page 102
- CTBCONPROPS on page 105

CTBCONFIG

Description

Sets or retrieves context properties.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONTEXT        FIXED BIN(31) INIT(0);
   01 RETCODE        FIXED BIN(31) INIT(0);
   01 ACTION         FIXED BIN(31);
   01 PROPERTY       FIXED BIN(31);
   01 BUFFER type;
   01 BUFFER_LEN  FIXED BIN(31);
   01 BUFBLANKSTRI   FIXED BIN(31);
   01 OUTLEN         FIXED BIN(31);

CALL CTBCONFI (CONTEXT, RETCODE, ACTION, PROPERTY, BUFFER,
BUFFER_LEN, BUFBLANKSTRIP, OUTLEN);
```

Parameters

CONTEXT

(I) A context structure. The context structure is defined in the program call CSBCTXALLOC. It corresponds to the *IHANDLE* structure in the Open ServerConnect Gateway-Library.

If this value is invalid or nonexistent, CTBCONFIG fails.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns” in this section.

ACTION

(I) Action to be taken by this call. *ACTION* is an integer variable that indicates the purpose of this call.

Assign *ACTION* one of the following symbolic values:

Value	Meaning
CS_GET (33)	Retrieves the value of the property.
CS_SET (34)	Sets the value of the property.
CS_CLEAR (35)	Clears the value of the property by resetting the property to its Client-Library default value.

PROPERTY

(I) Symbolic name of the property for which the value is being set or retrieved. Client-Library properties are listed under “Properties” on page 37, with description, possible values, and defaults.

BUFFER

(I/O) Variable (buffer) that contains the specified property value.

If *ACTION* is CS_SET, the buffer contains the value used by CTBCONFIG.

If *ACTION* is CS_GET, CTBCONFIG returns the requested information to this buffer.

If *ACTION* is CS_CLEAR, the buffer is reset to the default property value.

This argument is typically one of the following datatypes:

```
01 BUFFERFIXED BIN(n);
01 BUFFER  CHAR(n);
```

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If *ACTION* is CS_SET and the value in the buffer is a fixed-length or symbolic value, *BUFFER_LEN* should have a value of CS_UNUSED. To indicate that the terminating character is the last non-blank character, an application sets *BUFBLANKSTRIP* to CS_TRUE.

If *ACTION* is CS_GET and *BUFFER* is too small to hold the requested information, CTBCMDPROPS sets *OUTLEN* to the length of the requested information and returns CS_FAIL. To retrieve all the requested information, change the value of *BUFFER_LEN* to the length returned in *OUTLEN* and rerun the application.

If *ACTION* is CS_CLEAR, set this value to zeroes.

BUFBLANKSTRIP

(I) Blank stripping indicator. Indicates whether trailing blanks are stripped.

Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

OUTLEN

(O) Length, in bytes, of the retrieved information. *OUTLEN* is an integer variable where CTBCONFIG returns the length of the property value being retrieved.

When the retrieved information is larger than *BUFFER_LEN* bytes, an application uses the value of *OUTLEN* to determine how many bytes are needed to hold the information.

OUTLEN is used only when *ACTION* is CS_GET. If the *ACTION* is CS_SET or CS_CLEAR, this value is zero.

Return value

CTBCONFIG returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.

Examples

The following code fragment demonstrates how CTBCONFIG is used to determine the maximum number of connections. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*      ------------------------------------------------------*/
/*      find out what the maximum number of connections is    */
/*      ------------------------------------------------------*/
        CALL CTBCONFIG (CSL_CTX_HANDLE,
                        CSL_RC,
                        CS_GET,
                        CS_MAX_CONNECT,
                        CF_MAXCONNECT,
                        STG(CF_MAXCONNECT),
                        CS_FALSE,
                        CF_OUTLEN);

        .
        .  [check return code]
        .
        /* display number of connections */
        OR2_MAXCONNECT = CF_MAXCONNECT;

        TITLEO = OUTPUT_ROW_STR2;
```

Usage

- CSBCONFIG sets or retrieves the values of CS_EXTRA_INF and CS_VERSION. All other context properties are set or reset by CTBCONFIG. Context properties define aspects of Client-Library behavior at the context level.

- All connections created within a context pick up default property values from the parent context. An application can override these default values by calling CTBCONPROPS to set property values at the connection level.
- If an application changes context property values after allocating connections for the context, the existing connections do not recognize the new property values. Only new connections allocated after the new context property values are set use the new values as defaults.

See also

Related functions:

- CTBCMDPROPS on page 81
- CTBCONNECT on page 102
- CTBCONPROPS on page 105
- CTBINIT on page 147

Related topics:

- “Buffers” on page 21
- “Properties” on page 37

CTBCONNECT

Description

Connects to a server.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION          FIXED BIN(31) INIT(0);
   01 RETCODE             FIXED BIN(31) INIT(0);
   01 SERVERNAMECHAR(10) INIT('yourserver ')
   01 SERVERNAME_LEN   FIXED BIN(31);
   01 BUFBLANKSTRIP      FIXED BIN(31);

CALL CTBCONNE (CONNECTION, RETCODE, SERVERNAME,
SERVERNAME_LEN, BUFBLANKSTRIP);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

SERVERNAME

(I) Name of the connected server. For clients running SNA, this is the name by which the server is known to the Open ClientConnect Server Path Definition Table. For clients running TCP/IP without a gateway, this is the actual name of the Adaptive Server in the LAN interfaces file.

You must assign a value to this argument. If a server name is not specified, CTBCONNECT fails.

SERVERNAME_LEN

(I) Length, in bytes, of *SERVERNAME*. If the server name ends at the last non-blank character, assign CS_TRUE to *BUFBLANKSTRIP*.

BUFBLANKSTRIP

(I) Blank stripping indicator. Indicates whether trailing blanks are stripped.

Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

Return value

CTBCONNECT returns one of the following values listed in Table 3-8.

Table 3-8: CTBCONNECT return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_CRTABLE_UNAVAILABLE (-31)	The Connection Router table cannot be loaded.

Examples

The following code fragment demonstrates how to use CTBCONNECT. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------*/
/* open connection to the server or CICS region                */
/*-------------------------------------------------------------*/

         CALL CTBCONNE( CSL_CON_HANDLE,
                        CSL_RC,
                        PF_SERVER,
                        PF_SERVER_SIZE,
                        CS_FALSE ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR = 'CTBCONNECT failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

 /*-------------------------------------------------------------*/
 /* invokes SEND_COMMAND routine                                */
 /*-------------------------------------------------------------*/
         IF NO_ERRORS_SW
           THEN
             CALL SEND_COMMAND ;
```

Usage

- CTBCONNECT establishes a connection between a mainframe transaction processing region and a remote server. Information about the connection is stored in a connection handle, which uniquely identifies the connection.

- The remote server can be another transaction processing region or server (Adaptive Server, Open Server, and so on). For clients running SNA, the name in the Server Path Definition Table is the name of the remote region or server. For clients running TCP/IP, SYGWHOST (server name and IP address) is used in conjunction with the MVS-side information file for the specific drive or the CICS partner table.
- When it establishes a connection, CTBCONNECT sets up communication with the server, forwards login information, and communicates any connection-specific property information to the server.
- Because creating a connection involves sending login information, an application must define login parameters (server user ID and password) before calling CTBCONNECT. An application calls CTBCONPROPS to define login parameters.
- The maximum number of open connections per context is determined by the CS_MAX_CONNECT property (set by CTBCONFIG). The default maximum is 25 connections.
- There are two ways that an attempt to establish a connection can fail (assuming that the system is correctly configured):
  - If the specified server machine (the machine on which the server resides) is running correctly and the network is running correctly, but no server is listening on the specified port, the specified server machine signals the client, through a network error, that the connection cannot be formed. Regardless of the login time-out value, the connection fails.
  - If the machine on which the server resides is down, the server does not respond. Because “no response” is not considered to be an error, the network does not signal the client that an error occurred. However, if a login time-out period is set, a time-out error occurs when the client fails to receive a response within the set period.
- To close a connection, an application calls CTBCLOSE.

See also

Related functions:

- CTBCLOSE on page 72
- CTBCONALLOC on page 88
- CTBCONDROP on page 94
- CTBCONFIG on page 98
- CTBCONPROPS on page 105

- CTBREMOTEPWD on page 158

Related topics:

- "Properties" on page 37

CTBCONPROPS

Description

Sets or retrieves connection handle properties.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 CONNECTION       FIXED BIN(31) INIT(0);
    01 RETCODE          FIXED BIN(31) INIT(0);
    01 ACTION           FIXED BIN(31);
    01 PROPERTY         FIXED BIN(31);
    01 BUFFER type;
    01 BUFFER_LEN       FIXED BIN(31);
    01 BUFBLANKSTRIP    FIXED BIN(31);
    01 OUTLEN           FIXED BIN(31);

CALL CTBCONPR (CONNECTION, RETCODE, ACTION, PROPERTY,
BUFFER, BUFFER_LEN, BUFBLANKSTRIP, OUTLEN);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC. The connection handle corresponds to the *TDPROC* handle in the Open ServerConnect Gateway-Library.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

ACTION

(I) Action to be taken by this call. *ACTION* is an integer variable that indicates the purpose of this call.

Assign *ACTION* one of the following symbolic values:

Value	Meaning
CS_GET (33)	Retrieves the value of the property.
CS_SET (34)	Sets the value of the property.
CS_CLEAR (35)	Clears the value of the property by resetting the property to its Client-Library default value.

PROPERTY

(I) Symbolic name of the property for which the value is being set or retrieved. Client-Library properties are listed under "Properties" on page 37, with description, possible values, and defaults.

BUFFER

(I/O) Variable (buffer) that contains the specified property value:

- If *ACTION* is CS_SET, the buffer contains the value used by CTBCMDPROPS.
- If *ACTION* is CS_GET, CTBCMDPROPS returns the requested information to this buffer.
- If *ACTION* is CS_CLEAR, the buffer is reset to the default property value.

This argument is typically one of the following datatypes:

```
01 BUFFER FIXED BIN(n);
01 BUFFER CHAR(n);
```

BUFFER_LEN

(I/O) Length, in bytes, of the buffer.

If *ACTION* is CS_SET and the value in the buffer is a fixed-length or symbolic value, *BUFFER_LEN* should have a value of CS_UNUSED. To indicate that the terminating character is the last non-blank character, an application sets *BUFBLANKSTRIP* to CS_TRUE.

If *ACTION* is CS_GET and *BUFFER* is too small to hold the requested information, CTBCMDPROPS sets *OUTLEN* to the length of the requested information and returns CS_FAIL. To retrieve all the requested information, change the value of *BUFFER_LEN* to the length returned in *OUTLEN* and rerun the application.

BUFBLANKSTRIP

(I) Blank stripping indicator. Indicates whether trailing blanks are stripped.

Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

If you are setting a property value and the terminating character is the last non-blank character, assign CS_TRUE to *BUFBLANKSTRIP*.

OUTLEN

(O) Length, in bytes, of the retrieved information. *OUTLEN* is an integer variable where CTBCONPROPS returns the length of the property value being retrieved.

If the retrieved information is larger than *BUFFER_LEN* in bytes, an application uses the value of *OUTLEN* to determine how many bytes are needed to hold the information.

OUTLEN is used only when *ACTION* is CS_GET. If *ACTION* is CS_CLEAR or CS_SET, this value is zeroes.

Return value

CTBCONPROPS returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_CANNOT_SET_VALUE (-43)	This property cannot be set by the application.
TDS_INVALID_PARAMETER (-4)	One or more arguments contain illegal values.

Examples

The following code fragment demonstrates the use of CTBCONPROPS. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to process input data                                   */
/*                                                                    */
/*--------------------------------------------------------------------*/
 PROCESS_INPUT: PROC ;

 /*-------------------------------------------------------------*/
 /* allocate a connection to the server                         */
 /*-------------------------------------------------------------*/
         CSL_CON_HANDLE = 0 ;

         CALL CTBCONAL( CSL_CTX_HANDLE,
                        CSL_RC,
                        CSL_CON_HANDLE ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBCONALLOC failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
```

```
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for user-id              */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_USERNAME,
                       PF_USER,
                       PF_USER_SIZE,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for user-id failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for password             */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_PASSWORD,
                       PF_PWD,
                       PF_PWD_SIZE,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for password failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
```

```
/* alter properties of the connection for transaction         */
/*------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_TRANSACTION_NAME,
                       PF_TRAN,
                       PF_TRANL,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for transaction failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*------------------------------------------------------------*/
/* alter properties of the connection for Network driver      */
/*------------------------------------------------------------*/

        SELECT;
          WHEN (PF_NETDRV = '         ')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'LU62' | PF_NETDRV = 'lu62')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'IBMTCPIP' | PF_NETDRV = 'ibmtcpip')
                NETDRIVER = CS_TCPIP ;
          WHEN (PF_NETDRV = 'INTERLIN' | PF_NETDRV = 'interlin')
                NETDRIVER = CS_INTERLINK ;
          WHEN (PF_NETDRV = 'CPIC' | PF_NETDRV = 'cpic')
                NETDRIVER = CS_NCPIC ;
          OTHERWISE
            DO;
                MSGSTR = 'Invalid Network driver entered';
                NO_ERRORS_SW = FALSE ;
                CALL ERROR_OUT;
                CALL ALL_DONE ;
            END;
        END;

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
```

```
                    CS_SET,
                    CS_NET_DRIVER,
                    NETDRIVER,
                    CS_UNUSED,
                    CS_FALSE,
                    OUTLEN ) ;

IF CSL_RC ^= CS_SUCCEED THEN
DO ;
  MSGSTR = 'CTBCONPROPS for Network driver failed' ;
  NO_ERRORS_SW = FALSE ;
  CALL ERROR_OUT;
  CALL ALL_DONE ;
END ;
```

Usage

- CTBCONPROPS sets or retrieves the values of properties for a connection handle. Connection properties define aspects of Client-Library behavior at the connection level.
- All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling CTBCMDPROPS at the command structure level.
- If an application changes connection property values after allocating command structures for the connection, the existing command structures do not recognize the new property values. New command structures allocated for the connection use the new property values as defaults.
- Some connection properties only take effect if they are set before an application calls CTBCONNECT to establish the connection.
- An application can use CTBCONPROPS to set or retrieve the following properties:
  - CS_APPNAME
  - CS_CHARSETCNV
  - CS_COMMBLOCK
  - CS_EXTRA_INF
  - CS_HOSTNAME
  - CS_LOGIN_STATUS
  - CS_NET_DRIVER

- CS_NETIO
- CS_NOINTERRUPT
- CS_PACKETSIZE
- CS_PASSWORD
- CS_TDS_VERSION
- CS_TRANSACTION_NAME
- CS_USERDATA

See also

Related functions:

- CTBCMDPROPS on page 81
- CTBCONFIG on page 98
- CTBCONNECT on page 102
- CTBINIT on page 147

Related topics:

- “Buffers” on page 21
- “Properties” on page 37

CTBDESCRIBE

Description

Returns a description of result data.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);
   01 ITEM_NUM     FIXED BIN(31) INIT(1);
   01 DATAFMT,
      05 FMT_NAME       CHAR(132),
      05 FMT_NAMELEN    FIXED BIN(31),
      05 FMT_TYPE       FIXED BIN(31),
      05 FMT_FORMAT     FIXED BIN(31),
      05 FMT_MAXLEN     FIXED BIN(31),
      05 FMT_SCALE      FIXED BIN(31),
      05 FMT_PRECIS     FIXED BIN(31),
      05 FMT_STATUS     FIXED BIN(31),
      05 FMT_COUNT      FIXED BIN(31),
      05 FMT_UTYPE      FIXED BIN(31),
      05 FMT_LOCALE     FIXED BIN(31);

CALL CTBDESCR (COMMAND, RETCODE, ITEM_NUM, DATAFMT);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

ITEM_NUM

(I) Ordinal number of the column, parameter, or status being returned. This value is an integer.

When describing a column, *ITEM_NUM* is the column number. For example, the first column in the select list of a SQL select statement is column number 1, the second is column number 2, and so forth.

When describing a return parameter, *ITEM_NUM* is the ordinal rank of the parameter. The first parameter returned by a procedure or transaction is number 1. Adaptive Server stored procedure return parameters are returned in the order originally specified in the create procedure statement for the stored procedure. This is not necessarily the same order as specified in the RPC that invoked the stored procedure or transaction.

In determining what number to assign to *ITEM_NUM*, do not count non-return parameters. For example, if the second parameter in a stored procedure or transaction is the only return parameter, its *ITEM_NUM* is 1.

When describing a stored procedure return status, *ITEM_NUM* must be 1, because there can be only a single status in a return status result set.

To clear all bindings, assign *ITEM_NUM* a value of CS_UNUSED.

DATAFMT

(O) A structure that contains a description of the result data item referenced by *ITEM_NUM*. This structure is also used by CTBBIND, CTBPARAM and CSBCONVERT and is explained in the Topics chapter, under "DATAFMT structure" on page 26.

Warning! You must initialize *DATAFMT* to zeroes. Failure to do so causes addressing exceptions.

The *DATAFMT* structure contains the following fields listed in Table 3-9.

Table 3-9: DATAFMT return values

When this field	Is used with these result items	CTBDESCRIBE sets the field to
FMT_NAME	Regular columns, return parameters	The null-terminated name of the data item, if any. To indicate that there is no name, set FMT_NAMELEN to 0.
FMT_NAMELEN	Regular columns, return parameters	The actual length, in bytes, of FMT_NAME, not including the null terminator. A zero value here indicates no name.
FMT_TYPE	Regular columns, return parameters, return status	The datatype of the data item. All datatypes listed are valid. A return status always has a datatype of CS_INT.
FMT_FORMAT	Not used (CS_FMT_UNUSED)	Not applicable.
FMT_MAXLEN	Regular columns, return parameters	The maximum possible length, in bytes, of the data for the column or parameter being described.
FMT_SCALE	Regular columns and return parameters for which the datatype is packed decimal (CS_PACKED370), or Sybase-decimal/numeric	The number of digits to the right of the decimal point.
FMT_PRECIS	Regular columns and return parameters for which the datatype is packed decimal (CS_PACKED370), or Sybase-decimal/numeric	The total number of decimal digits in the result data item.

When this field	Is used with these result items	CTBDESCRIBE sets the field to
FMT_STATUS	Regular columns only	One or more of the following symbolic values, added together: • CS_CANBENULL to indicate a column that was tagged “nullable” by the server. • CS_NODATA to indicate that no data is associated with the column.
FMT_COUNT	Regular columns, return parameters, return status	The number of rows copied to destination variables per CTBFETCH call. CTBDESCRIBE initializes FMT_COUNT as 1 to provide a default value in case an application uses the CTBDESCRIBE return DATAFMT structure as the CTBBIND input DATAFMT structure. This value is always 1 for return parameters and status results.
FMT_UTYPE	Regular columns, return parameters	The user-defined datatype of the column or parameter, if any. FMT_UTYPE is set in addition to (not instead of) DATATYPE. **Note** This field is used for datatypes defined at the server, not for Open Client user-defined datatypes.
FMT_LOCALE	Reserved for future use (CS_FORMAT_UNUSED)	Reserved for future use.

Return value

CTBDESCRIBE returns one of the following values listed in Table 3-10.

Table 3-10: CTBDESCRIBE return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. CTBDESCRIBE returns CS_FAIL if *ITEM_ NUM* does not represent a valid result data item.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_TERMINATED (-4997)	The connection is not active.
TDS_NO_COMPUTES_ALLOWED (-60)	Compute results are not supported.
TDS_RESULTS_CANCELED (-49)	A cancel was sent to purge results.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call.

Examples

The following code fragment demonstrates the use of CTBDESCRIBE. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process result rows                                 */
/*                                                                   */
/*-------------------------------------------------------------------*/
 RESULT_ROW_PROCESSING: PROC ;

/*-----------------------------------------------------------*/
/* We need to bind the data to program variables. We don't   */
/* care about the indicator variable so we'll pass NULL for  */
/* that parameter in CTBBIND().                              */
/*-----------------------------------------------------------*/

         CALL CTBRESIN( CSL_CMD_HANDLE,
                        CSL_RC,
                        CS_NUMDATA,
                        RF_NUMDATA,
                        STG(RF_NUMDATA),
                        CF_COL_LEN ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBRESINFO failed' ;
           NO_ERRORS_SW = FALSE ;
```

```
            CALL ERROR_OUT;
            CALL ALL_DONE;
          END ;

          FF_ROW_NUM = FF_ROW_NUM + 1;

/*-------------------------------------------------------------*/
/* display the number of connections                           */
/*-------------------------------------------------------------*/

          OR2_MAXCONNECT     = CF_MAXCONNECT ;
          RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR2 ;
          FF_ROW_NUM         = FF_ROW_NUM + 2;

 /*-------------------------------------------------------------*/
 /* display the number of columns                               */
 /*-------------------------------------------------------------*/

          OR4_NUMDATA        = RF_NUMDATA ;
          RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR4 ;

          IF RF_NUMDATA ^= 2 THEN
          DO ;
            MSGSTR       = 'CTBRESINFO returned wrong # of parms' ;
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
          END ;

          FF_ROW_NUM = FF_ROW_NUM + 2;

 /*-------------------------------------------------------------*/
 /* Setup column headings                                       */
 /*-------------------------------------------------------------*/

          RSLTNO(FF_ROW_NUM) = 'FirstName    EducLvl' ;
          FF_ROW_NUM         = FF_ROW_NUM + 1;
          RSLTNO(FF_ROW_NUM) = '===========  =======' ;

          DO PARM_CNT = 1 TO RF_NUMDATA  ;
            CALL BIND_COLUMNS ;
          END ;

 END RESULT_ROW_PROCESSING ;

 /*-------------------------------------------------------------------*/
```

```
/*                                                                    */
/* Subroutine to bind each data                                       */
/*                                                                    */
/*--------------------------------------------------------------------*/
BIND_COLUMNS: PROC ;

        CALL CTBDESCR( CSL_CMD_HANDLE,
                       CSL_RC,
                       PARM_CNT,
                       DATAFMT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBDESCRIBE failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

/*--------------------------------------------------------------*/
/* We need TO bind the data TO program variables. We don't      */
/* care about the indicator variable so we'll pass NULL for     */
/* that PARAMeter in OC_BIND().                                 */
/*--------------------------------------------------------------*/

/*--------------------------------------------------------------*/
/* rows per fetch                                               */
/*--------------------------------------------------------------*/

        DF_COUNT = 1 ;

        SELECT( DF_DATATYPE ) ;
/*--------------------------------------------------------------*/
/* bind the first column, FIRSTNME defined as VARCHAR(12)       */
/*--------------------------------------------------------------*/
          WHEN( CS_VARCHAR_TYPE )
          DO ;
            DF_DATATYPE   = CS_VARCHAR_TYPE;
            DF_FORMAT     = CS_FMT_UNUSED;
            DF_MAXLENGTH  = STG(CF_COL_FIRSTNME) - 2;
            DF_COUNT      = 1;
            CF_COL_NUMBER = 1;

            CALL CTBBIND( CSL_CMD_HANDLE,
                          CSL_RC,
                          CF_COL_NUMBER,
```

```
                              DATAFMT,
                              CF_COL_FIRSTNME,
                              CF_COL_LEN,
                              CS_PARAM_NOTNULL,
                              CF_COL_INDICATOR,
                              CS_PARAM_NULL);

                IF CSL_RC ^= CS_SUCCEED THEN
                DO ;
                  MSGSTR       = 'CTBBIND CS_VARCHAR_TYPE failed' ;
                  NO_ERRORS_SW = FALSE ;
                  CALL ERROR_OUT;
                  CALL ALL_DONE;
                END ;
              END ;

  /*-------------------------------------------------------------*/
  /* bind the second column, EDLEVEL defined as SMALLINT         */
  /*-------------------------------------------------------------*/
              WHEN( CS_SMALLINT_TYPE )
              DO ;
                DF_DATATYPE    = CS_SMALLINT_TYPE;
                DF_FORMAT      = CS_FMT_UNUSED;
                DF_MAXLENGTH   = STG(CF_COL_EDLEVEL);
                DF_COUNT       = 1;
                CF_COL_NUMBER = 2;

                CALL CTBBIND( CSL_CMD_HANDLE,
                              CSL_RC,
                              CF_COL_NUMBER,
                              DATAFMT,
                              CF_COL_EDLEVEL,
                              CF_COL_LEN,
                              CS_PARAM_NOTNULL,
                              CF_COL_INDICATOR,
                              CS_PARAM_NULL ) ;

                IF CSL_RC ^= CS_SUCCEED THEN
                DO ;
                  MSGSTR       = 'CTBBIND CS_SMALLINT_TYPE failed' ;
                  NO_ERRORS_SW = FALSE ;
                  CALL ERROR_OUT;
                  CALL ALL_DONE;
                END ;
              END ;
```

```
            OTHERWISE ;

        END ; /* end of SELECT( DF_DATATYPE ) */

END BIND_COLUMNS ;
```

Usage

- CTBDESCRIBE returns a complete description of a result data item in the current result set. Result data items include regular result columns, return parameters, and stored procedure return status values.
- An application can call CTBRESINFO to find out how many result items are present in the current result set.
- An application generally needs to call CTBDESCRIBE to describe a result data item after it establishes a connection and sends a request, and before it binds the result item to a program variable using CTBBIND.

See also

Related functions:

- CTBBIND on page 59
- CTBFETCH on page 139
- CTBRESINFO on page 161
- CTBRESULTS on page 167

Related topics:

- “DATAFMT structure” on page 26
- “Results” on page 47

CTBDIAG

Description

Manages in-line error handling.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 CONNECTION  FIXED BIN(31) INIT(0);
    01 RETCODE     FIXED BIN(31) INIT(0);
    01 COMPILER    FIXED BIN(31);
    01 OPERATION   FIXED BIN(31);
    01 MSGTYPE     FIXED BIN(31);
    01 INDEX       FIXED BIN(31) INIT(1);
```

```
01 BUFFER type;

CALL CTBDIAG (CONNECTION, RETCODE, COMPILER, OPERATION,
MSGTYPE, INDEX, BUFFER);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

COMPILER

This argument is ignored.

OPERATION

(I) Operation to perform. Assign this argument one of the following values:

Value	Meaning
CS_GET (33)	Retrieves a specific message.
CS_CLEAR (35)	Clears message information for this connection.
CS_INIT (36)	Initializes in-line error handling.
CS_STATUS (37)	Returns the current number of stored messages.
CS_MSGLIMIT (38)	Sets the maximum number of messages to store.

MSGTYPE

(I) Type of message or structure on which the operation is to be performed. Assign this argument one of the following symbolic values:

CS_CLIENTMSG_TYPE (4700)	A CLIENTMSG structure. Indicates Client-Library messages.
CS_SERVERMSG_TYPE (4701)	A SERVERMSG structure. Indicates messages sent by the Mainframe ClientConnect or other server.
CS_ALLMSG_TYPE (4702)	Operation is performed on both Client-Library and server messages.
SQLCA_TYPE (4703)	A SQLCA structure.
SQLCODE_TYPE (4704)	A SQLCODE structure.

INDEX

(I) Index number of the message being retrieved. Messages are numbered sequentially: the first message has an index of 1, the second an index of 2, and so forth.

- If *MSGTYP* is CS_CLIENTMSG_TYPE, then *INDEX* refers to Client-Library messages only.
- If *MSGTYP* is CS_SERVERMSG_TYPE, then *INDEX* refers to server messages only.
- If *MSGTYP* is CS_ALLMSG_TYPE, then *INDEX* refers to both Client-Library and server messages.
- *INDEX* should be initialized to 1.

BUFFER

(I/O) An integer or a variable ("buffer") that contains the message. See Table 3-11, to learn the relationship between *BUFFER* and other arguments.

This argument is typically either CHAR, a SQLCA structure, or a CLIENTMSG or SERVERMSG structure.

Note It is the responsibility of the programmer to provide a buffer large enough to hold the largest possible message.

Return value

CTBDIAG returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. Common reasons for a CTBDIAG failure include: • Invalid *CONNECTION*. • Inability to allocate memory. • Invalid parameter (for example, parameter is not allowed for operation). • Invalid parameter combination.
CS_NOMSG (-207)	The application attempted to retrieve a message for which the index number is greater than the number of messages in the queue. For example, the application attempted to retrieve message number 3, when only 2 messages are queued.

Examples

Example 1

The following example uses CTBDIAG to prepare to receive messages. This example is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process input data                                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
 PROCESS_INPUT: PROC ;

/*-----------------------------------------------------------*/
/* allocate a connection to the server                       */
/*-----------------------------------------------------------*/

         CSL_CON_HANDLE = 0 ;

         CALL CTBCONAL( CSL_CTX_HANDLE,
                        CSL_RC,
                        CSL_CON_HANDLE ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBCONALLOC failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

 /*-----------------------------------------------------------*/
 /* alter properties of the connection for user-id            */
 /*-----------------------------------------------------------*/

         CALL CTBCONPR( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_SET,
                        CS_USERNAME,
                        PF_USER,
                        PF_USER_SIZE,
                        CS_FALSE,
                        OUTLEN ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR = 'CTBCONPROPS for user-id failed' ;
           NO_ERRORS_SW = FALSE ;
```

```
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for password             */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_PASSWORD,
                       PF_PWD,
                       PF_PWD_SIZE,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for password failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for transaction          */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_TRANSACTION_NAME,
                       PF_TRAN,
                       PF_TRANL,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for transaction failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;
```

```
/*-------------------------------------------------------------*/
/* alter properties of the connection for Network driver       */
/*-------------------------------------------------------------*/

        SELECT;
          WHEN (PF_NETDRV = '        ')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'LU62' | PF_NETDRV = 'lu62')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'IBMTCPIP' | PF_NETDRV = 'ibmtcpip')
                NETDRIVER = CS_TCPIP ;
          WHEN (PF_NETDRV = 'INTERLIN' | PF_NETDRV = 'interlin')
                NETDRIVER = CS_INTERLINK ;
          WHEN (PF_NETDRV = 'CPIC' | PF_NETDRV = 'cpic')
                NETDRIVER = CS_NCPIC ;
          OTHERWISE
            DO;
                MSGSTR = 'Invalid Network driver entered';
                NO_ERRORS_SW = FALSE ;
                CALL ERROR_OUT;
                CALL ALL_DONE ;
            END;
        END;

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_NET_DRIVER,
                       NETDRIVER,
                       CS_UNUSED,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for Network driver failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* setup retrieval of All Messages                             */
/*-------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
```

```
                        CSL_RC,
                        CS_UNUSED,
                        CS_INIT,
                        CS_ALLMSG_TYPE,
                        CS_UNUSED,
                        CS_UNUSED ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_INIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* set the upper limit of number of messages                   */
/*-------------------------------------------------------------*/

        PF_MSGLIMIT = 5 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_MSGLIMIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      PF_MSGLIMIT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_MSGLIMIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;
/*-------------------------------------------------------------*/
/* open connection to the server or CICS region                */
/*-------------------------------------------------------------*/
   CALL CTBCONNE( CSL_CON_HANDLE,
                  CSL_RC,
                  PF_SERVER,
                  PF_SERVER_SIZE,
                  CS_FALSE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
```

```
        DO ;
          MSGSTR = 'CTBCONNECT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

 /*-------------------------------------------------------------*/
 /* invokes SEND_COMMAND routine                                */
 /*-------------------------------------------------------------*/
        IF NO_ERRORS_SW
          THEN
            CALL SEND_COMMAND ;
```

Example 2

The following example uses CTBDIAG to retrieve diagnostic messages. This example is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to retrieve any diagnostic messages                    */
/*                                                                   */
/*-------------------------------------------------------------------*/
 GET_DIAG_MESSAGES: PROC ;

 DCL CNT        FIXED BIN(15) ;

/*-------------------------------------------------------------------*/
/* Disable calls to this subroutine                                  */
/*-------------------------------------------------------------------*/

        DIAG_MSGS_INITIALIZED = FALSE ;

/*-------------------------------------------------------------------*/
/* First, get client messages                                        */
/*-------------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_STATUS,
                      CS_CLIENTMSG_TYPE,
                      CS_UNUSED,
                      DF_NUM_OF_MSGS ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
```

```
          DO ;
            MSGSTR = 'CTBDIAG CS_STATUS CLIENTMSG_TYPE failed';
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;
          ELSE DO ;
            IF DF_NUM_OF_MSGS > 0 THEN
              DO ;
                DO CNT = 1 TO DF_NUM_OF_MSGS ;
                  CALL RETRIEVE_CLIENT_MSGS ;
                END ;
              END ;
          END ;

/*-------------------------------------------------------------------*/
/* Then, get server messages                                         */
/*-------------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_STATUS,
                      CS_SERVERMSG_TYPE,
                      CS_UNUSED,
                      DF_NUM_OF_MSGS ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_STATUS SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;
          ELSE DO ;
            IF DF_NUM_OF_MSGS > 0 THEN
              DO ;
                DO CNT = 1 TO DF_NUM_OF_MSGS ;
                  CALL RETRIEVE_SERVER_MSGS ;
                END ;
              END ;
          END ;

END GET_DIAG_MESSAGES ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to retrieve diagnostic messages from client            */
```

```
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_CLIENT_MSGS: PROC ;

        I1 = 1 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
                      CS_CLIENTMSG_TYPE,
                      DF_MSGNO,
                      CLIENT_MSG ) ;

        IF CSL_RC ^=  CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_GET CS_CLIENTMSG_TYPE FAILED' ;
          CALL ERROR_OUT ;
          CALL ALL_DONE ;
        END ;

/*--------------------------------------------------------------------*/
/* display message text                                               */
/*--------------------------------------------------------------------*/

        RSLTNO( I1 )      = 'Client Message:' ;
        I1                = 3 ;

        CM_SEVERITY_DATA = CM_SEVERITY ;
        CM_STATUS_DATA   = CM_STATUS ;
        RSLTNO( I1 )      = CM_SEVERITY_HDR || CM_SEVERITY_DATA ||
                            CM_STATUS_HDR   || CM_STATUS_DATA ;
        I1                = I1 + 1 ;

        CM_OC_MSGNO_DATA = CM_MSGNO ;
        RSLTNO( I1 )      = CM_OC_MSGNO_HDR || CM_OC_MSGNO_DATA ;
        I1                = I1 + 1 ;

        IF CM_MSGNO ^= 0 THEN
        DO ;
            CM_OC_MSG_DATA     = SUBSTR( CM_TEXT, 1, 66 ) ;
            RSLTNO( I1 )       = '  OC MsgTx:  ' || CM_OC_MSG_DATA ;
            I1                 = I1 + 1 ;
            IF CM_TEXT_LEN > 66 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 67, 66 ) ;
```

```
            RSLTNO( I1 )      = BLANK_13 || CM_OC_MSG_DATA_X ;
            I1                = I1 + 1 ;
            IF CM_TEXT_LEN > 132 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 133, 66 ) ;
                RSLTNO( I1 )      = BLANK_13 ||
                                    CM_OC_MSG_DATA_X ;
                I1                = I1 + 1 ;
                IF CM_TEXT_LEN > 198 THEN
                DO ;
                    CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 199 ) ;
                    RSLTNO( I1 )      = BLANK_13 ||
                                        CM_OC_MSG_DATA_X ;
                    I1                = I1 + 1 ;
                END ;
            END ;
        END ;
    END ;
    ELSE DO ;
      RSLTNO( I1 ) = '  OC MsgTx:  No Message!' ;
      I1           = I1 + 1 ;
    END ;

    CM_OS_MSGNO_DATA = CM_OS_MSGNO ;
    RSLTNO( I1 )      = '  OS MsgNo:  ' || CM_OS_MSGNO_DATA ;
    I1                = I1 + 1 ;

    IF CM_OS_MSGNO ^= 0 THEN
    DO ;
        CM_OS_MSG_DATA     = SUBSTR( CM_OS_MSGTXT, 1, 66 ) ;
        RSLTNO( I1 )       = '  OS MsgTx:  ' ||
                             CM_OS_MSG_DATA ;
        I1                 = I1 + 1 ;
        IF CM_OS_MSGTEXT_LEN > 66 THEN
        DO ;
            CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT, 67, 66 ) ;
            RSLTNO( I1 )      = BLANK_13 || CM_OC_MSG_DATA_X ;
            I1                = I1 + 1 ;
            IF CM_OS_MSGTEXT_LEN > 132 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,133,66 );
                RSLTNO( I1 )      = BLANK_13 || CM_OC_MSG_DATA_X ;
                I1                = I1 + 1 ;
                IF CM_OS_MSGTEXT_LEN > 198 THEN
                DO ;
                    CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,199 );
```

```
                        RSLTNO( I1 )      = BLANK_13 ||
                                            CM_OC_MSG_DATA_X ;
                        I1                = I1 + 1 ;
                    END ;
                END ;
            END ;
        END ;
        ELSE DO ;
          RSLTNO( I1 ) = '  OS MsgTx:  No Message!' ;
          I1           = I1 + 1 ;
        END ;

END RETRIEVE_CLIENT_MSGS ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve diagnostic messages from server             */
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_SERVER_MSGS: PROC ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
                      CS_SERVERMSG_TYPE,
                      DF_MSGNO,
                      SERVER_MSG ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_GET CS_SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;

/*--------------------------------------------------------------------*/
/* display message text                                               */
/*--------------------------------------------------------------------*/
        SM_MSG_NO_DATA     = SM_MSGNO ;
        SM_SEVERITY_DATA   = SM_SEV ;
        SM_STATE_DATA      = SM_STATE ;
        SM_LINE_NO_DATA    = SM_LINE ;
        SM_STATUS_DATA     = SM_STATUS ;

        IF SM_SVRNAME_LEN > 66
```

```
            THEN
              SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 63 ) || '...' ;
            ELSE
              SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 66 ) ;

          IF SM_PROC_LEN > 66
            THEN
              SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 63 ) || '...' ;
            ELSE
              SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 66 ) ;

          SM_MSG_DATA         = SUBSTR( SM_TEXT, 1, 66 ) ;
          RSLTNO (1)          = 'Server Message:' ;
          RSLTNO (3)          = SM_MSG_NO_HDR   || SM_MSG_NO_DATA ||
                                SM_SEVERITY_HDR || SM_SEVERITY_DATA ||
                                SM_STATE_HDR    || SM_STATE_DATA ;
          RSLTNO (4)          = SM_LINE_NO_HDR  || SM_LINE_NO_DATA ||
                                SM_STATUS_HDR   || SM_STATUS_DATA ;
          RSLTNO (5)          = SM_SVRNAME_HDR  || SM_SVRNAME_DATA ;
          RSLTNO (6)          = SM_PROC_ID_HDR  || SM_PROC_ID_DATA ;
          RSLTNO (7)          = SM_MSG_HDR      || SM_MSG_DATA ;

          IF SM_TEXT_LEN > 66 THEN
            DO ;
              SM_MSG_DATA_X = SUBSTR( SM_TEXT, 67, 66 ) ;
              RSLTNO(8)     = BLANK_13 || SM_MSG_DATA_X ;
              IF SM_TEXT_LEN > 132 THEN
                DO ;
                  SM_MSG_DATA_X = SUBSTR( SM_TEXT, 133, 66 ) ;
                  RSLTNO(9)     = BLANK_13 || SM_MSG_DATA_X ;
                  IF SM_TEXT_LEN > 198 THEN
                    DO ;
                      SM_MSG_DATA_X = SUBSTR( SM_TEXT, 198 ) ;
                      RSLTNO(10)    = BLANK_13 || SM_MSG_DATA_X ;
                    END ;
                END ;
            END ;
    END RETRIEVE_SERVER_MSGS ;
```

Usage

- CTBDIAG manages in-line message handling for a specific connection. If an application has more than one connection, it must make separate CTBDIAG calls for each connection.
- Open ClientConnect applications always use CTBDIAG to handle Client-Library and server messages. Applications built with Open Client can provide alternative message-handling facilities.

- An application can perform operations on Client-Library messages, server messages, or both.

  For example, an application can clear Client-Library messages without affecting server messages:

  ```
  CALL CTBDIAG (CONNECTION,RETCODE, CS_UNUSED,
  CS_CLEAR, CS_CLIENTMSG, CS_UNUSED, MSGBUFFER);
  ```

- CTBDIAG allows an application to retrieve message information into standard Client-Library structures (CLIENTMSG, SERVERMSG, SQLCA or SQLCODE.

- When retrieving messages, CTBDIAG assumes that *BUFFER* points to a structure of the type indicated by *MSGTYPE*.

  - An application that is retrieving messages into a SQLCA or SQLCODE structure must set the Client-Library property CS_EXTRA_INF to CS_TRUE. This is because the SQL structures require information that is not ordinarily returned by the Client-Library error handling mechanism.

    Use CTBCONPROPS or CSBCONFIG to set CS_EXTRA_INF.

  - An application that does not use the SQLCA or SQLCODE structures can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard Client-Library messages.

  For more information about CS_EXTRA_INF, see "Properties" on page 37 and the reference pages for CTBCONPROPS and CSBCONFIG.

Warning! If CTBDIAG does not have sufficient internal storage space in which to save a new message, it throws away all unread messages and stops saving messages. The next time it is called with *OPERATION* as CS_GET, it returns a message to indicate the space problem.
After returning this message, CTBDIAG starts saving messages again.

Initializing in-line error handling

An application must initialize in-line error handling before it can retrieve any errors. To initialize in-line error handling, call CTBDIAG with *OPERATION* as CS_INIT.

- Generally, if a connection uses in-line error handling, the application should call CTBDIAG to initialize in-line error handling for a connection immediately after allocating it with CTBCONALLOC.

Clearing messages

To clear message information for a connection, an application calls *OPERATION* as CS_CLEAR.

- To clear Client-Library messages only, set *MSGTYPE* to CS_CLIENTMSG_TYPE.
- To clear server messages only, set *MSGTYPE* to CS_SERVERMSG_TYPE.
- To clear both Client-Library and server messages, set *MSGTYPE* to SQLCA, SQLCODE, or CS_ALLMSG_TYPE.
- If *OPERATION* is CS_CLEAR and *MSGTYPE* is not CS_ALLMSG_TYPE:
  - CTBDIAG assumes that *BUFFER* is a structure of type *MSGTYPE*.
  - CTBDIAG clears the buffer by setting it to blanks or zeroes, as appropriate.
- Message information is not cleared until an application explicitly calls CTBDIAG with *OPERATION* as CS_CLEAR. Retrieving a message does not remove it from the message queue.

Retrieving messages

To retrieve message information, an application calls CTBDIAG with *OPERATION* as CS_GET, *MSGTYPE* as the type of structure in which to retrieve the message, *INDEX* as the index number of the message of interest, and *BUFFER* as an integer or a variable, as appropriate.

- If *MSGTYPE* is CS_CLIENTMSG_TYPE, *INDEX* refers only to Client-Library messages.
- If *MSGTYPE* is CS_SERVERMSG_TYPE, *INDEX* refers only to server messages.
- If *MSGTYPE* has any other value, *INDEX* refers to the collective “queue” of both types of messages combined.
- CTBDIAG creates a messages queue in the buffer and fills the buffer with message information. It returns messages to the client in the order in which they are received.
- If an application attempts to retrieve a message with an index that is higher than the highest valid index, CTBDIAG returns CS_NOMSG to indicate that a message is not available.

Limiting messages

The Client-Library default behavior is to save an unlimited number of messages. Applications running on platforms with limited memory may want to limit the number of messages that Client-Library saves. The default for MVS is 25.

- An application can limit the number of saved Client-Library messages, the number of saved server messages, and the total number of saved messages.
- To limit the number of saved messages, an application calls CTBDIAG with *OPERATION* as CS_MSGLIMIT and *MSGTYPE* as CS_CLIENTMSG_TYPE, CS_SERVERMSG_TYPE, or CS_ALLMSG_TYPE:
  - If *MSGTYPE* is CS_CLIENTMSG_TYPE, then the number of Client-Library messages is limited.
  - If *MSGTYPE* is CS_SERVERMSG_TYPE, then the number of server messages is limited.
  - If *MSGTYPE* is CS_ALLMSG_TYPE, then the total number of Client-Library and server messages combined is limited.
- When a specific message limit is reached, Client-Library discards any new messages of that type. When a combined message limit is reached, Client-Library discards any new messages.

Retrieving the number of messages

To find out how many messages were retrieved, an application calls CTBDIAG with *OPERATION* as CS_STATUS and *MSGTYPE* as the type of message of interest.

Table 3-11 on page 134 provides a summary of CTBDIAG arguments.

Table 3-11: Summary of CTBDIAG arguments

OPERATION	CTBDIAG action	MSGTYPE value	INDEX value	BUFFER value
CS_INIT	Initializes in-line error handling. An application must call CTBDIAG with a CS_INIT operation before it can process error messages in line.	CS_UNUSED	CS_UNUSED	Ignored.

OPERATION	CTBDIAG action	MSGTYPE value	INDEX value	BUFFER value
CS_CLEAR	Clears message information for this connection. If *BUFFER* is not zeroes and *MSGTYPE* is not CS_ALLMSG_TYPE, CTBDIAG clears the buffer by initializing it with blanks or zeroes.	One of the legal *MSGTYPE* values. • If *MSGTYPE* is CS_CLIENTMSG_TYPE, CTBDIAG clears Client-Library messages only. • If *MSGTYPE* is CS_SERVERMSG_TYPE, CTBDIAG clears server messages only. • If *MSGTYPE* has any other legal value, CTBDIAG clears both Client-Library and server messages.	CS_UNUSED	A buffer for which the type is defined by *MSGTYPE*.
CS_GET	Retrieves a specific message.	Any legal *MSGTYPE* value except CS_ALLMSG_TYPE. • If *MSGTYPE* is CS_CLIENTMSG_TYPE, CTBDIAG retrieves a Client-Library message into a CLIENTMSG structure. • If *MSGTYPE* is CS_SERVERMSG_TYPE, CTBDIAG retrieves a server message into a SERVERMSG structure. • If *MSGTYPE* has any other value, CTBDIAG retrieves either a server message or a Client-Library message.	The index number of the message to retrieve.	A buffer for which the type is defined by *MSGTYPE*.

OPERATION	CTBDIAG action	MSGTYPE value	INDEX value	BUFFER value
CS_MSGLIMIT	Sets the maximum number of messages to store.	CS_CLIENTMSG_TYPE to limit Client-Library messages only. CS_SERVERMSG_TYPE to limit server messages only. CS_ALLMSG_TYPE to limit the total number of Client-Library and server messages combined.	CS_UNUSED	An integer value.
CS_STATUS	Returns the current number of stored messages.	CS_CLIENTMSG_TYPE to retrieve the number of Client-Library messages. CS_SERVERMSG_TYPE to retrieve the number of server messages. CS_ALLMSG_TYPE to retrieve the total number of Client-Library and server messages combined.	CS_UNUSED	An integer variable.

See also

Related topics:

- “Error and message handling” on page 33
- “CLIENTMSG structure” on page 24
- “SERVERMSG structure” on page 48
- “SQLCA structure” on page 50

CTBEXIT

Description

Exits Client-Library.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONTEXT     FIXED BIN(31) INIT(0);
   01 RETCODE     FIXED BIN(31) INIT(0);
   01 OPTION      FIXED BIN(31);
```

```
CALL CTBEXIT (CONNECTION, RETCODE, OPTION);
```

Parameters

CONTEXT

(I) A context structure. The context structure is defined in the program call CSBCTXALLOC. This value identifies the Client-Library context being exited.

If this value is invalid or nonexistent, CTBEXIT fails.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

OPTION

(O) Indicator specifying whether or not CTBEXIT closes connections for which results are pending.

CTBEXIT behaves in differently, depending on the value specified for *OPTION*. The following table lists the symbolic values that are legal for *OPTION*.

OPTION value	CTBEXIT action
CS_UNUSED (-99999)	Closes all open connections for which no results are pending and terminates Client-Library for this context. If results are pending for one or more connections, CTBEXIT returns CS_FAIL and does not terminate Client-Library.
CS_FORCE_EXIT (300)	Closes all open connections for this context, whether or not any results are pending, and terminates Client-Library for this context.

Return value

CTBEXIT returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_INVALID_PARAMETER (-4)	A parameter contains an illegal value.
TDS_RESULTS_STILL_ACTIVE (-50)	Some results are still pending.
TDS_ CONNECTION_ TERMINATED (-4997)	The connection is not active.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call.

Examples

The following code fragment demonstrates the use of CTBEXIT. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
   /*                                                                 */
   /* Subroutine to perform exit client library and deallocate context */
   /* structure.                                                      */
   /*                                                                 */
   /*-------------------------------------------------------------------*/
   QUIT_CLIENT_LIBRARY: PROC ;

   /*-------------------------------------------------------------------*/
   /* exit the Client Library                                         */
   /*-------------------------------------------------------------------*/
         CALL CTBEXIT( CSL_CTX_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;
        IF CSL_RC = CS_FAIL THEN
       DO ;
         MSGSTR = 'CTBEXIT failed' ;
         CALL ERROR_OUT ;
       END ;
   /*-------------------------------------------------------------------*/
   /* de-allocate the context stricture                               */
   /*-------------------------------------------------------------------*/
       CALL CSBCTXDR( CSL_CTX_HANDLE,
                      CSL_RC ) ;
        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CSBCTXDROP failed' ;
          CALL ERROR_OUT ;
        END ;
EXEC CICS RETURN ;
       END QUIT_CLIENT_LIBRARY ;
```

Usage

- CTBEXIT terminates a Client-Library context. It closes all open connections, deallocates internal data space and cleans up any platform-specific initialization.
- CTBEXIT must be the last Client-Library routine called within a Client-Library context.
- If an application needs to call Client-Library routines after it calls CTBEXIT, it can re-initialize Client-Library by calling CTBINIT again.

- If results are pending on any of the context connections and *OPTION* is not passed as CS_FORCE_EXIT, CTBEXIT returns CS_FAIL. This means that Client-Library is not correctly terminated. The application must handle the pending results before calling CTBEXIT, or it can call CTBEXIT again, specifying CS_ FORCE_EXIT.
- To close a single connection, an application calls CTBCLOSE.
- If CTBINIT is called for a context, the application must call CTBEXIT before it calls CSBCTXDROP to deallocate the context.

See also

Related functions:

- CTBCLOSE on page 72

CTBFETCH

Description

Fetches result data.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 COMMAND       FIXED BIN(31) INIT(0);
    01 RETCODE       FIXED BIN(31) INIT(0);
    01 TYP           FIXED BIN(31);
    01 OFFSET        FIXED BIN(31);
    01 OPTION        FIXED BIN(31);
    01 ROWS_READ     FIXED BIN(31);

 CALL CTBFETCH (COMMAND, RETCODE, TYP, OFFSET, OPTION,
ROWS_READ);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Return value,” in this section.

TYP

(I) This argument is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.

OFFSET

(I) This argument is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.

OPTION

(I) This argument is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.

ROWS_READ

(I) Variable where the number of result rows is returned. This variable is of type integer. CTBFETCH sets ROWS_ READ to the number of rows read by the CTBFETCH call. This argument is required.

Return value

CTBFETCH returns one of the following values listed in Table 3-12.

Table 3-12: CTBFETCH return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully. CTBFETCH places the total number of rows read in ROWS_READ.
CS_FAIL (-2)	The routine failed. CTBFETCH places the number of rows fetched before the failure occurred in ROWS_READ. A common reason for a CTBFETCH failure is that a program variable specified through CTBBIND is too small to hold a fetched data item.
CS_CANCELLED (-202)	The operation was canceled. CTBFETCH places the number of rows fetched before the cancel occurred in ROWS_READ.
CS_ROW_FAIL (-203)	A recoverable error occurred while fetching a row. Recoverable errors include memory allocation failures and conversion errors that occur while copying row values to program variables. An application can continue calling CTBFETCH to continue retrieving rows, or can call CTBCANCEL to cancel the remaining results. CTBFETCH places the number of rows fetched before the error occurred in ROWS_READ, then continues by fetching the row after the error.
CS_END_DATA (-204)	No more rows are available in this result set. (Note that this is also a successful completion.)
TDS_INVALID_PARAMETER (-4)	One of the CTBFETCH arguments contains an illegal value. The most likely cause of this code is assigning a value other than CS_UNUSED to one of more of the reserved arguments, *TYP*, *OFFSET*, and *OPTION*.
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call. It is in Send state instead of Receive state.

Examples

The following example shows a typical use of CTBFETCH. It is taken from the SYCTSAA4 sample program in Appendix A, “Sample Language Application.”

```
   /*-------------------------------------------------------------------*/
   /*                                                                   */
   /* Subroutine to fetch row processing                                */
   /*                                                                   */
   /*-------------------------------------------------------------------*/
      FETCH_ROW_PROCESSING: PROC ;

        CALL CTBFETCH( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_UNUSED,       /* type   */
                       CS_UNUSED,       /* offset */
                       CS_UNUSED,       /* option */
                       FF_ROWS_READ ) ;

        SELECT( CSL_RC ) ;

          WHEN( CS_SUCCEED )
          DO ;
            NO_MORE_ROWS         = FALSE ;
            CF_COL_FIRSTNME_CHAR = BLANK ;
            DF_DATATYPE          = CS_VARCHAR_TYPE;
            DF_MAXLENGTH         = LENGTH( CF_COL_FIRSTNME ) ;
            DF2_DATATYPE         = CS_CHAR_TYPE;
            DF2_MAXLENGTH        = STG(CF_COL_FIRSTNME_CHAR);

            CALL CSBCONVE( CSL_CTX_HANDLE,
                           CSL_RC,
                           DATAFMT,
                           CF_COL_FIRSTNME,
                           DATAFMT2,
                           CF_COL_FIRSTNME_CHAR,
                           CF_COL_LEN);

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CSCONVERT CS_CHAR_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
            END ;

            FF_ROW_NUM = FF_ROW_NUM + 1 ;

/*-------------------------------------------------------------*/
/* save ROW RESULTS for later display                          */
/*-------------------------------------------------------------*/
```

```
            OR_COL_FIRSTNME_CHAR = CF_COL_FIRSTNME_CHAR;
            OR_COL_EDLEVEL       = CF_COL_EDLEVEL;

            IF FF_ROW_NUM > 10 THEN
            DO;
              MSG_TEXT_1 = 'Please press return to continue!' ;
              MSG_TEXT_2 = BLANK ;
              CALL DISP_DATA ;
              FF_ROW_NUM = 1;
              PAGE_CNT = PAGE_CNT + 1 ;

/*-------------------------------------------------------------*/
/* Setup column headings                                       */
/*-------------------------------------------------------------*/

              RSLTNO(FF_ROW_NUM) = 'FirstName    EducLvl' ;
              FF_ROW_NUM         = FF_ROW_NUM + 1 ;
              RSLTNO(FF_ROW_NUM) = '===========  =======' ;
              FF_ROW_NUM         = FF_ROW_NUM + 1 ;
            END ;

            RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR;

          END ; /* end of WHEN( CS_SUCCEED ) */

          WHEN( CS_END_DATA )
          DO ;
            NO_MORE_ROWS = TRUE ;
            MSG_TEXT_1   = 'All rows processing completed!' ;
            MSG_TEXT_2   = 'Press Clear To Exit';
            CALL DISP_DATA ;
          END ; /* end of WHEN( CS_END_DATA ) */

          WHEN( CS_FAIL )
          DO ;
            NO_MORE_ROWS = TRUE ;
            NO_ERRORS_SW = FALSE ;
            MSGSTR       =
                'CTBFETCH returned CS_FAIL ret_code' ;
            CALL ERROR_OUT;
          END ; /* end of WHEN( CS_FAIL ) */

          WHEN( CS_ROW_FAIL )
          DO ;
            NO_MORE_ROWS = TRUE ;
```

```
                NO_ERRORS_SW = FALSE ;
                MSGSTR       =
                    'CTBFETCH returned CS_ROW_FAIL ret_code' ;
                CALL ERROR_OUT;
              END ; /* end of WHEN( CS_ROW_FAIL ) */

              WHEN( CS_CANCELLED )
              DO ;
                NO_MORE_ROWS = TRUE ;
                NO_ERRORS_SW = FALSE ;
                MSG10     = 'CTBFETCH returned CS_CANCELLED ret_code' ;
                CALL ERROR_OUT;
              END ; /* end of WHEN( CS_CANCELLED ) */
          OTHERWISE
              DO ;
                NO_MORE_ROWS = TRUE ;
                NO_ERRORS_SW = FALSE ;
                MSGSTR       =
                    'CTBFETCH returned Unknown ret_code' ;
                CALL ERROR_OUT;
              END ; /* end of OTHERWISE */
    END ; /* end of SELECT( CSL_RC ) */
    END FETCH_ROW_PROCESSING ;
```

Usage

- CTBFETCH fetches result data. “Result data” is an umbrella term for the various types of data that a server can return to an application. These types of data include:
  - Regular rows
  - Return parameters, including both message parameters and RPC return parameters
  - Stored procedure status results

  CTBFETCH is used to fetch all of these types of data.
- Conceptually, result data is returned to an application in the form of one or more rows that make up a “result set.”

  Regular row result sets can contain more than one row. For example, a regular row result set might contain a hundred rows. If array binding is specified for the data items in a regular row result set, then multiple rows can be fetched with a single call to CTBFETCH. The number of rows fetched are returned in the *ROWS_READ* argument.

Return parameters and status results, however, only contain a single row. For this reason, even if array binding is specified, only a single row of data is fetched.

- CTBRESULTS specifies the type of result available in the RESULT_TYP variable. CTBRESULTS must indicate a result type of CS_ROW_RESULT, CS_PARAM_RESULT, or CS_STATUS_RESULT before an application calls CTBFETCH.
- After calling CTBRESULTS, an application can:
  - Process the result set by binding the result items and fetching the data, using CTBFETCH (optionally preceded by CTBDESCRIBE and CTBBIND), or
  - Discard the result set, using CTBCANCEL.
- If an application does not cancel a result set, it must completely process the result set by repeatedly calling CTBFETCH as long as CTBFETCH continues to indicate that rows are available.

  The simplest way to do this is in a loop that terminates when CTBFETCH fails to return either CS_SUCCEED or CS_ROW_FAIL. After the loop terminates, an application can check the CTBFETCH final return code to find out what caused the termination.

Fetching regular rows

- Regular rows can be fetched from the server one row at a time, or several rows at once.
- When fetching multiple rows, the number of rows to be fetched is indicated by the FMT_COUNT field in the DATAFMT structures used to bind the data items in the result set. Note that the FMT_COUNT field must have the same value for all CTBBIND calls for a result set.

  If FMT_COUNT is 0 or 1, CTBFETCH fetches one row.

Fetching return parameters

- A return parameter result set contains either stored procedure return parameters or message parameters.
- A return parameter result set consists of a single row with a number of columns equal to the number of return parameters.

Fetching a return status

- A stored procedure return status result set consists of a single row with a single column, containing the status.

See also

Related functions:

- CTBBIND on page 59
- CTBDESCRIBE on page 112
- CTBRESULTS on page 167

CTBGETFORMAT

Description

Returns the user-defined format for a result column.

Note This function is used with requests to Adaptive Server Enterprise only.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND       FIXED BIN(31) INIT(0);
   01 RETCODE       FIXED BIN(31) INIT(0);
   01 COLUMN_NUM    FIXED BIN(31) INIT(1);
   01 BUFFER        FIXED BIN(31);
   01 BUFFER_LEN    FIXED BIN(31);
   01 OUTLEN        FIXED BIN(31);

CALL CTBGETFO (COMMAND, RETCODE, COLUMN_NUM, BUFFER,
BUFFER_LEN, OUTLEN);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

COLUMN_NUM

(I) Number of the column for which the user-specified format is desired.

COLUMN_NUM refers to the select-list ID of the column. The first column in the select list of a select statement is column number 1, the second is column number 2, and so forth.

BUFFER

(O) Variable ("buffer") in which CTBGETFORMAT places the requested information.

This argument is typically:

```
01 BUFFER CHAR(n);
```

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If *BUFFER_LEN* is too small to hold the requested information, CTBGETFORMAT sets *OUTLEN* to the length of the requested information, and returns CS_FAIL.

OUTLEN

(O) Length, in bytes, of the format string. *OUTLEN* is an integer variable where CTBGETFORMAT returns the total number of bytes being retrieved.

When the format string is larger than *BUFFER_LEN* bytes, an application uses this value to determine how many bytes are needed to hold the string.

If a format string is not associated with the specified column, CTBGETFORMAT sets *OUTLEN* to 0.

Return value

CTBGETFORMAT returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_INVALID_PARAMETER (-4)	One of the CTBGETFORMAT arguments contains an illegal value.

Usage

- CTBGETFORMAT returns the user-defined format, if any, for a result column. It indicates how the field should be formatted on screen.
- An application can call CTBGETFORMAT after CTBRESULTS indicates results of type CS_ROW_RESULT.
- For a description of how to add user-defined formats to Adaptive Server databases or Open Servers, see the Adaptive Server and Open Server documentation.

See also

Related functions:

- CTBBIND on page 59
- CTBDESCRIBE on page 112

CTBINIT

Description

Initializes Client-Library.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 CONTEXT      FIXED BIN(31) INIT(0);
    01 RETCODE      FIXED BIN(31) INIT(0);
    01 VERSION      FIXED BIN(31);

 CALL CTBINIT (CONNECTION, RETCODE, VERSION);
```

Parameters

CONTEXT

(I) A context structure. The context structure is defined in the program call CSBCTXALLOC. If this value is invalid or nonexistent, CTBINIT fails.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

VERSION

(I) Version of Client-Library behavior that the application expects. The following table lists the symbolic values that are legal for *VERSION*:

Value	Meaning	Supported features
CS_VERSION_46	Application communicates with a version 4.6 Adaptive Server.	RPCs.
CS_VERSION_50	Application communicates with a version 10.0 SQL Server and above.	RPCs.

Return value

CTBINIT returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_MEM_ERROR (-3)	The routine failed due to a memory allocation error.
CS_FAIL (-2)	The routine failed for other reasons. **Note** CTBINIT returns CS_FAIL if Client-Library cannot provide version-level behavior.
TDS_INVALID_PARAMETER (-4)	A parameter contains an illegal value. The most likely cause is an erroneous version number.

Value	Meaning
TDS_WRONG_STATE (-6)	Program is in the wrong communication state to issue this call. The most likely cause is that this context already initiated.

Examples

The following code fragment demonstrates how CTBINIT is used with other functions to initialize a program. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/* program initialization                                             */
/*--------------------------------------------------------------------*/

        DIAG_MSGS_INITIALIZED = TRUE ;
        MSG_TEXT_2            = 'Press Clear To Exit';
        NO_ERRORS_SW          = TRUE ;
        PAGE_CNT              = PAGE_CNT + 1;
        SERVERL               = -1 ;

        DO I1 = 1 TO 13 ;
          RSLTNO( I1 ) = BLANK ;
        END ;

        CALL GET_SYSTEM_TIME ;

GET_INPUT_AGAIN:

          CALL DISPLAY_INITIAL_SCREEN ;
          CALL GET_INPUT_DATA ;

/*--------------------------------------------------------------------*/
/* allocate a context structure                                       */
/*--------------------------------------------------------------------*/

        CALL CSBCTXAL( CS_VERSION_50,
                       CSL_RC,
                       CSL_CTX_HANDLE );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CSCTXALLOC failed';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;
```

```
/*---------------------------------------------------------------------*/
/* initialize the Client-Library                                       */
/*---------------------------------------------------------------------*/

        CALL CTBINIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_VERSION_50 );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CTBINIT failed';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;

        CALL PROCESS_INPUT ;

        CALL QUIT_CLIENT_LIBRARY ;
```

Usage

- CTBINIT initializes Client-Library. It sets up internal control structures and defines the version of Client-Library behavior that an application expects. Client-Library provides the requested behavior, regardless of the actual version of Client-Library in use.
- CTBINIT must be the first Client-Library routine call after CSBCTXALLOC. Other Client-Library routines fail if they are called before CTBINIT.
- Because an application calls CTBINIT before it sets up error handling, an application must check the CTBINIT return code to detect failure.
- It is not an error for an application to call CTBINIT multiple times. Some applications cannot guarantee which of several modules executes first. In such a case, each module should contain a call to CTBINIT.

See also

Related functions:

- CSBCTXALLOC on page 190
- CTBEXIT on page 136

CTBPARAM

Description

Defines a command parameter.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND           FIXED BIN(31) INIT(0);
   01 RETCODE           FIXED BIN(31) INIT(0);
   01 DATAFMT,
      05 FMT_NAME       CHAR(132),
      05 FMT_NAMELEN    FIXED BIN(31),
      05 FMT_TYPE       FIXED BIN(31),
      05 FMT_FORMAT     FIXED BIN(31),
      05 FMT_MAXLEN     FIXED BIN(31),
      05 FMT_SCALE      FIXED BIN(31),
      05 FMT_PRECIS     FIXED BIN(31),
      05 FMT_STATUS     FIXED BIN(31),
      05 FMT_COUNT      FIXED BIN(31),
      05 FMT_UTYPE      FIXED BIN(31),
      05 FMT_LOCALE     FIXED BIN(31);
   01 DATA         type;
   01 DATALEN           FIXED BIN(31);
   01 INDICATOR         FIXED BIN(15) INIT(0);

CALL CTBBIND (COMMAND, RETCODE, DATAFMT, DATA, DATALEN,
INDICATOR);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

DATAFMT

(I) A structure that contains a description of the parameter. This structure is also used by CTBBIND, CTBDESCRIBE and CSBCONVERT and is explained in “DATAFMT structure” on page 26.

Table 3-13 on page 151 lists the fields in the DATAFMT structure, indicates whether or when they are used by CTBPARAM, and contains general information about the fields.

Note The programmer is responsible for adhering to these rules. Client-Library does not enforce them.

Table 3-13: Fields in the DATAFMT structure for CTBPARAM

When this field	Is used in this condition	Set the field to
FMT_NAME	When defining parameters for all supported commands.	The name of the parameter being defined. If FMT_NAMELEN is 0, the parameter is considered to be unnamed. Unnamed parameters are interpreted positionally. It is an error to mix named and unnamed parameters in a single command. **Note** When sending parameters to an Adaptive Server, FMT_NAME must begin with the "@" symbol, which prefixes all Adaptive Server stored procedure parameter names. When sending parameters with language requests, this must be the variable name as it appears in the language string. Transact-SQL names begin with the colon (:) symbol.
FMT_NAMELEN	When defining parameters for all supported commands.	The length, in bytes, of FMT_NAME. If FMT_NAMELEN is 0, the parameter is considered to be unnamed.
FMT_TYPE	When defining parameters for all supported commands.	The datatype of the parameter value. All datatypes listed under "Datatypes" on page 30 are valid.
FMT_FORMAT	Not used (CS_FMT_UNUSED).	Not applicable.
FMT_MAXLEN	When defining non-fixed-length return parameters for RPCs; otherwise CS_UNUSED.	The maximum length, in bytes, of the data returned in this parameter. For character or binary data, FMT_MAXLEN must represent the total length of the return parameter, including any space required for special terminating bytes, with this exception: when the parameter is a VARYCHAR datatype such as the DB2 VARCHAR, FMT_MAXLEN does not include the length of the "LL" length specification. For Sybase-decimal and Sybase-numeric, set FMT_MAXLEN to 35. If the parameter is non-return, if FMT_TYPE is fixed-length, or if the application does not need to restrict the length of return parameters, set FMT_MAXLEN to CS_UNUSED.
FMT_SCALE	Used for packed decimal, Sybase-decimal, and Sybase-numeric datatypes.	The number of digits after the decimal point.

When this field	Is used in this condition	Set the field to
FMT_PRECIS	Used for packed decimal, Sybase-decimal, and Sybase-numeric datatypes.	The total number of digits before and after the decimal point.
FMT_STATUS	When defining parameters for all types of commands except message commands.	The type of parameter being defined. One of the following values: • CS_INPUTVALUE - The parameter is an input parameter value for a non-return RPC parameter or a language request parameter. • CS_RETURN - The parameter is a return parameter.
FMT_COUNT	Not used (CS_FMT_UNUSED).	Not applicable.
FMT_UTYPE	Only when defining a parameter that has an Adaptive Server user-defined datatype; otherwise CS_UNUSED.	The user-defined datatype of the parameter, if any. FMT_UTYPE is set in addition to (not instead of) DATATYPE. **Note** This field is used for datatypes defined at the server, not for Open Client user-defined datatypes.
FMT_LOCALE	Not used (CS_FMT_UNUSED).	Zeroes.

DATA

Variable that contains the parameter data.

To indicate a null parameter (zeroes), assign *INDICATOR* a value of -1.

If *INDICATOR* is -1, *DATA* and *DATA_LEN* are ignored. For example, an application might pass empty parameters to a stored procedure or transaction that assigns default values to empty input parameters.

DATA_LEN

The length, in bytes, of the parameter data. For Sybase-numeric and Sybase-decimal, set *DATA_LEN* to 35.

INDICATOR

An integer variable used to indicate an empty parameter. To indicate that a parameter is empty, assign *INDICATOR* a value of -1. If *INDICATOR* is -1, *DATA* and *DATA_LEN* are ignored.

Return value

CTBPARAM returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.

Value	Meaning
CS_FAIL (-2)	The routine failed.

Examples

The following code fragment illustrates the use of CTBPARAM. It is taken from the sample program SYCTSAR4 in Appendix B, "Sample RPC Application."

```
/*---------------------------------------------------------------*/
/* prepare the command (an RPC request)                          */
/*---------------------------------------------------------------*/
        PF_STRLEN = STG(CF_CMD);

        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_RPC_CMD,
                       CF_CMD,
                       PF_STRLEN,
                       CS_UNUSED );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCOMMAND failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*---------------------------------------------------------------*/
/*                                                               */
/* setup a return parameter for NUM_OF_ROWS                      */
/*                                                               */
/* describe the first parameter (NUM_OF_ROWS)                    */
/*                                                               */
/*---------------------------------------------------------------*/

        DF_NAME         = '@parm1';
        DF_NAMELEN      = 6;
        DF_DATATYPE     = CS_INT_TYPE;
        DF_FORMAT       = CS_FMT_UNUSED;
        DF_MAXLENGTH    = CS_UNUSED;
        DF_STATUS       = CS_RETURN;
        DF_USERTYPE     = CS_UNUSED;

        PM_LEN          = STG(PM_PARAM1);
        PM_PARAM1       = 0;                    /* NUM_OF_ROWS */
        PM_NULLIND      = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
```

```
                        CSL_RC,
                        DATAFMT,
                        PM_PARAM1,
                        PM_LEN,
                        PM_NULLIND );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBPARAM CS_INT_TYPE parm1 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/*                                                             */
/* describe the second parameter (DEPTNO)                      */
/*                                                             */
/*-------------------------------------------------------------*/

        DF_NAME         = '@parm2';
        DF_NAMELEN      = 6;
        DF_DATATYPE     = CS_VARCHAR_TYPE;
        DF_FORMAT       = CS_FMT_UNUSED;
        DF_MAXLENGTH    = CS_UNUSED;
        DF_STATUS       = CS_INPUTVALUE;
        DF_USERTYPE     = CS_UNUSED;

        PM_PARAM2       = PF_DEPT;                  /* DEPTNO */
        PM_LEN          = PF_DEPT_SIZE ;
        PM_NULLIND      = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
                       CSL_RC,
                       DATAFMT,
                       PM_PARAM2,
                       PM_LEN,
                       PM_NULLIND );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBPARAM CS_VARCHAR_TYPE parm2 failed' ;
          CALL ERROR_OUT;
```

```
            CALL ALL_DONE ;
          END ;

    /*-------------------------------------------------------------*/
    /* send the command                                            */
    /*-------------------------------------------------------------*/

          CALL CTBSEND( CSL_CMD_HANDLE,
                        CSL_RC ) ;

          IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR       = 'CTBSEND failed' ;
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE ;
          END ;

    END SEND_PARAM ;
```

Usage

- An application calls CTBCOMMAND to initiate a language request, RPC or message command.
- An application calls CTBPARAM once for each parameter that is sent with the current RPC. It describes each parameter. That description is forwarded to the procedure or transaction called.
- CTBPARAM defines parameters for the following types of commands:
  - Language requests
  - RPCs
- A language request requires input parameter values when the text of the language request contains host variables.
- Parameters must be described by CTBPARAM in the same order in which they are sent to the server. The first CTBPARAM call describes the first parameter, the second CTBPARAM call describes the second parameter, and so on, until all parameters are described and sent.

Defining arguments for language requests

- An application calls CTBPARAM with FMT_STATUS as CS_INPUTVALUE to define a parameter value for a language request containing variables.
- A language request can have up to 255 parameters.

- Table 3-14 lists the fields in the DATAFMT structure that take special values when describing a parameter for a language request.

Table 3-14: DATAFMT structure language requests to CTBPARAM

Field	Value
NAME	The variable name as it appears in the language string. Transact-SQL names begin with the colon (:) character.
FMT_STATUS	CS_INPUTVALUE
All other fields	Standard CTBPARAM values.

Defining arguments for RPCs

- An application calls CTBPARAM with FMT_STATUS as CS_RETURN to define a return parameter for an RPC, and calls CTBPARAM with FMT_STATUS as CS_INPUTVALUE to define a non-return parameter.
- An application can call a stored procedure or transaction in two ways: (1) by sending a language request or (2) by issuing an RPC. See "Remote procedure calls (RPCs)" on page 44 for a discussion of the differences between these techniques.
- To send an RPC, a Client-Library application:
  - Calls CTBCOMMAND to initiate the request.
  - Calls CTBPARAM once for each parameter that is being passed to the remote procedure.
  - Calls CTBSEND to send the request to the server. One CTBSEND forwards the RPC with all defined parameters; the application does not call CTBSEND separately for each parameter.
- An RPC can have up to 255 parameters.
- The following fields in the DATAFMT structure listed in Table 3-15 take special values when describing an RPC parameter.

Table 3-15: DATAFMT fields for RPC parameters with CTBPARAM

Field	Value
FMT_NAME	When sending parameters to an Adaptive Server, FMT_NAME must begin with the "@" symbol, which prefixes all Adaptive Server stored procedure parameter names.
FMT_MAXLEN	The maximum length of data to be returned by the server. Set to CS_UNUSED if the parameter is non-return, if FMT_TYPE is fixed-length, or if the application does not need to restrict the length of return parameters.
FMT_STATUS	CS_RETURN to indicate that the parameter is a return parameter. CS_INPUTVALUE to indicate that the parameter is not a return parameter.

Field	Value
All other fields	Standard CTBPARAM values.

Table 3-16 lists a summary of CTBPARAM arguments.

Table 3-16: Summary of arguments (CTBPARAM)

Command	FMT_STATUS value	DATA, DATA_LEN value
Language request	CS_INPUTVALUE	The parameter value and length.
RPC (return parameters)	CS_RETURN	The parameter value and length.
RPC (non-return parameters)	CS_INPUTVALUE	The parameter value and length.

See also

Related functions:

- CTBCOMMAND on page 84
- CTBSEND on page 174

Related topics:

- “DATAFMT structure” on page 26

CTBREMOTEPWD

Description

Defines or clears passwords to be used for server-to-server connections.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONNECTION    FIXED BIN(31) INIT(0);
   01 RETCODE       FIXED BIN(31) INIT(0);
   01 ACTION        FIXED BIN(31) INIT(1);
   01 SERVERNAMECHAR(30);
   01 SRV_LEN       FIXED BIN(31);
   01 SRV_BLANKSTRIP    FIXED BIN(31);
   01 PASSWDCHAR(30);
   01 PWD_LEN       FIXED BIN(31);
   01 PWD_BLANKSTRIP    FIXED BIN(31);

CALL CTBREMOT (CONNECTION, RETCODE, ACTION, SERVERNAME,
SRV_LEN, SRV_BLANKSTRIP, PASSWD, PWD_LEN, PWD_BLANKSTRIP);
```

Parameters

CONNECTION

(I) Handle for this connection. This connection handle must already be allocated with CTBCONALLOC.

Remote passwords can only be defined for a connection before it is open. Passwords defined after a connection is open are ignored.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

ACTION

(I) Action to be taken by this call. *ACTION* is an integer variable that indicates the purpose of this call. *ACTION* can be any of the following symbolic values:

Value	Meaning
CS_SET (34)	Sets the remote password.
CS_CLEAR (35)	Clears all remote passwords specified for this connection.

SERVERNAME

(I) Name of the server for which the password is being defined. This is the name by which the server is known in the Server Path Table.

If *ACTION* is CS_CLEAR, set *SERVERNAME* to zeroes.

If *SERVERNAME* is zeroes, the specified password will be considered a “universal” password, to be used with any server that does not have a password explicitly specified for it.

SERVERNAME_LEN

(I) Length, in bytes, of *SERVERNAME*. To use the default "universal" password, assign CS_ NULL_ STRING to this argument. To indicate that the value is terminated at the last non-blank character, assign CS_TRUE to *SRVBLANKSTRIP*.

SRVBLANKSTRIP

(I) Blank stripping indicator. Indicates whether the value in the buffer is terminated at the last non-blank character. Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

PASSWD

(I) Password being installed for remote logins to the server named in *SERVERNAME*.

If ACTION is CS_CLEAR, set *PASSWD* to zeroes, and the password defaults to the one set for this connection in CTBCONPROPS, if any.

PASSWD_LEN

(I) Length, in bytes, of *PASSWD*. To indicate that the value is terminated at the last non-blank character, assign CS_TRUE to *PWDBLANKSTRIP*.

PWDBLANKSTRIP

(I) Blank termination indicator. Indicates whether the value of the password is terminated at the last non-blank character.

Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

Return value

CTBREMOTEPWD returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	Results are available for processing.
CS_FAIL (-2)	The routine failed.

Value	Meaning
TDS_INVALID_PARAMETER (-4)	One or more of the CTBREMOTEPWD arguments contains an illegal value. Likely causes for this code are: • Erroneous value for *ACTION*. *ACTION* cannot be CS_GET for CTBREMOTEPWD. • Erroneous value for a length argument. Length values cannot be negative numbers.
TDS_SOS (-257)	Memory shortage. The operation failed.

Usage

- CTBREMOTEPWD defines the password that a server uses when it logs into another server. An application can call CTBREMOTEPWD to clear remote passwords for a connection at any time.
- A Transact-SQL language command, stored procedure, or transaction running on one server can call a stored procedure or transaction located on another server. To accomplish this server-to-server communication, the first server, to which an application connected through CTBCONNECT, actually logs in to the second, remote server, performing a server-to-server remote procedure call.

  CTBREMOTEPWD allows an application to specify the password to be used when the first server logs in to the remote server.
- Multiple passwords can be specified, one for each server that the first server might need to log in to. Each password must be defined with a separate call to CTBREMOTEPWD.
- If an application does not specify a remote password for a particular server, the password defaults to the password set for this connection through CTBCONPROPS, if any. If a password is not defined, the password is set to zeroes. If an application user generally has the same password on different servers, this default behavior can be sufficient.
- Remote passwords are stored in an internal buffer, which is only 255 bytes long. Each password entry in the buffer consists of the password itself, the associated server name, and two extra bytes. If the addition of a password to this buffer would cause overflow, CTBREMOTEPWD returns CS_FAIL and generates a Client-Library error message that indicates the problem.
- Define remote passwords before calling CTBCONNECT to create an active connection. It is an error to call CTBREMOTEPWD to define a remote password for a connection that is already open.

See also

Related functions:

- CTBCONNECT on page 102
- CTBCONPROPS on page 105

CTBRESINFO

Description

Returns result set information.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);
   01 RESULT_TYP   FIXED BIN(31);
   01 BUFFER       type;
   01 BUFFER_LEN   FIXED BIN(31);
   01 OUTLEN       FIXED BIN(31);

CALL CTBRESIN (COMMAND, RETCODE, RESULT_TYP, BUFFER,
BUFFER_LEN, OUTLEN);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

RESULT_TYP

(I) Type of information to return. Assign this argument one of the following values:

Value	Meaning
CS_ROW_COUNT (800)	The number of rows affected by the current command.
CS_CMD_NUMBER (801)	The number of the command that generated the current result set.
CS_NUMDATA (803)	The number of items in the current result set.

BUFFER

(O) Variable (“buffer”) where CTBRESINFO returns the requested information. At present, this will always be an integer value.

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If the returned value is longer than *BUFFER_LEN*, CTBRESINFO sets *OUTLEN* to the length of the requested information and returns CS_FAIL.

OUTLEN

(O) Length, in bytes, of the retrieved information. *OUTLEN* is an integer variable where CTBRESINFO returns the length of the information being retrieved.

If the retrieved information is larger than *BUFFER_LEN* bytes, an application uses the value of *OUTLEN* to determine how many bytes are needed to hold the information.

Return value

CTBRESINFO returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	Results are available for processing.
CS_FAIL (-2)	The routine failed. CTBRESINFO returns CS_FAIL if the requested information is larger than BUFFER_LEN bytes.

Examples

The following code fragment demonstrates the use of CTBRESINFO. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to process result rows                                  */
/*                                                                    */
/*--------------------------------------------------------------------*/
RESULT_ROW_PROCESSING: PROC ;

 /*-------------------------------------------------------------*/
 /* We need to bind the data to program variables. We don't     */
 /* care about the indicator variable so we'll pass NULL for    */
 /* that parameter in OC_BIND().                                */
 /*-------------------------------------------------------------*/

         CALL CTBRESIN( CSL_CMD_HANDLE,
                        CSL_RC,
                        CS_NUMDATA,
                        RF_NUMDATA,
                        STG(RF_NUMDATA),
                        CF_COL_LEN ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
```

```
        DO ;
          MSGSTR       = 'CTBRESINFO failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

        FF_ROW_NUM = FF_ROW_NUM + 1;

 /*-------------------------------------------------------------*/
 /* display the number of connections                           */
 /*-------------------------------------------------------------*/

        OR2_MAXCONNECT     = CF_MAXCONNECT ;
        RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR2 ;
        FF_ROW_NUM         = FF_ROW_NUM + 2;

/*-------------------------------------------------------------*/
/* display the number of columns                               */
/*-------------------------------------------------------------*/

        OR4_NUMDATA        = RF_NUMDATA ;
        RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR4 ;

        IF RF_NUMDATA ^= 2 THEN
        DO ;
          MSGSTR       = 'CTBRESINFO returned wrong # of parms' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

        FF_ROW_NUM = FF_ROW_NUM + 2;

 /*-------------------------------------------------------------*/
 /* Setup column headings                                       */
 /*-------------------------------------------------------------*/

        RSLTNO(FF_ROW_NUM) = 'FirstName    EducLvl' ;
        FF_ROW_NUM         = FF_ROW_NUM + 1;
        RSLTNO(FF_ROW_NUM) = '===========  =======' ;

        DO PARM_CNT = 1 TO RF_NUMDATA  ;
          CALL BIND_COLUMNS ;
        END ;
```

```
END RESULT_ROW_PROCESSING ;
```

Usage

- CTBRESINFO returns information about the current result set or the current command. The current command is defined as the request that generated the current result set.
- A result set is a collection of a single type of result data. Result sets are generated by requests. For more information on result sets, see CTBRESULTS on page 167.

Retrieving the command number for the current result set

- To determine the number of the command that generated the current result set, call CTBRESINFO with *RESULT_TYP* as CS_CMD_NUMBER.
- Client-Library keeps track of the command number by counting the number of times CTBRESULTS returns CS_CMD_DONE.

  An application's first call to CTBRESULTS following a CTBSEND call sets the command number to 1. The command number remains 1 until CTBRESULTS returns CS_CMD_DONE. The next time the application calls CTBRESULTS, the command number is incrementally increased to 2. The command number continues to increase by 1 each time CTBRESULTS is called after returning CS_CMD_DONE.
- CS_CMD_NUMBER is useful in the following cases:
  - To determine the SQL command within a language request that generated the current result set.
  - To determine the select command in a stored procedure or transaction that generated the current result set.
- A language request contains a string of text. This text represents one or more SQL commands or other language request statements. If the application is sending a language request, "command number" refers to the number of the statement in the language request.

  For example, the following Transact-SQL string represents three Transact-SQL commands—two select statements and one insert:

```
select * from authors
select * from titles
insert newauthors
select * from authors
where city = "San Francisco"
```

The two select statements can generate result sets. In this case, the command number that CTBRESINFO returns can be from 1 to 3, depending on when CTBRESINFO is called.

Note When sending SQL strings to DB2, remember to use semicolons (;) to separate SQL statements.

- Inside stored procedures or transactions, only select statements cause the command number to be incremented. If a stored procedure or transaction contains seven SQL commands, three of which are select statements, the command number that CTBRESINFO returns can be any integer from 1 to 3, depending on which select statement generated the current result set.

Retrieving the number of result data items

- To determine the number of result data items in the current result set, call CTBRESINFO with *RESULT_TYP* as CS_NUMDATA.
- Results sets contain result data items. Row result sets contain columns, a parameter result set contains parameters, and a status result set contains a status. The columns, parameters, and status are known as result data items.

Retrieving the number of rows for the current command

- To determine the number of rows affected by the current command, call CTBRESINFO with *RESULT_TYP* as CS_ROW_COUNT.
- If the current command is one that does not return rows—for example, a language command containing an insert statement—an application can get the row count immediately after CTBRESULTS returns CS_CMD_SUCCEED.
- If the current command does return rows:
  - An application can get a total row count after processing all of the rows.
  - An application can get an intermediate row count any time after CTBRESULTS indicates that results are available. An intermediate row count is equivalent to the number of rows that have been fetched so far.

- If the command is one that executes a stored procedure or transaction—such as a Transact-SQL exec language command or a remote procedure call—CTBRESINFO returns either the number of rows returned by the latest select statement executed by the stored procedure or transaction, or CS_NO_COUNT if the stored procedure or transaction does not execute any select statements. A stored procedure or transaction that does not contain any select statements can execute a select by calling another stored procedure or transaction that contains a select statement.
- CTBRESINFO returns CS_NO_COUNT if any of the following are true:
  - The SQL command fails for any reason, such as a syntax error.
  - The command is one that *never* affects rows, such as a Transact-SQL print command.
  - The command executes a stored procedure or transaction that does not execute any select statements.

The following arguments listed in Table 3-17 are returned to *RESULT_TYP* after CTBRESULTS indicates that results are present.

Table 3-17: Summary of arguments (CTBRESINFO)

RESULT_TYP	CTBRESINFO returns	BUFFER value
CS_ROW_COUNT (800)	The number of rows affected by the current command.	An integer value.
CS_CMD_NUMBER (801)	The number of the command that generated the current result set.	An integer value.
CS_NUMDATA (803)	The number of items in the current result set.	An integer value.

See also

Related functions:

- CTBCMDPROPS on page 81
- CTBCONPROPS on page 105
- CTBRESULTS on page 167

Related topics:

- “Results” on page 47

CTBRESULTS

Description Sets up result data to be processed.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 COMMAND      FIXED BIN(31) INIT(0);
    01 RETCODE      FIXED BIN(31) INIT(0);
    01 RESULT_TYP   FIXED BIN(31);
 CALL CTBRESUL (COMMAND, RETCODE, RESULT_TYP);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

RESULT_TYP

(O) Variable containing the result type. CTBRESULTS returns to this variable a symbolic value that indicates the type of result returned by the current request. The result type can be any of the following symbolic values listed in Table 3-18.

Table 3-18: Values for RESULT_TYP (for CTBRESULTS)

Value	Meaning	Result produced
CS_ROW_RESULT (4040)	Regular row results arrived.	One or more rows of tabular data.
CS_PARAM_RESULT (4042)	Return parameter results arrived.	A single row of return parameters.
CS_STATUS_RESULT (4043)	Stored procedure return status results arrived.	A single row containing a single status.
CS_CMD_DONE (4046)	The results of the request processed completely.	Not applicable.
CS_CMD_SUCCEED (4047)	A request that returns no data, such as a language request containing an insert statement, processed successfully.	No results.
CS_CMD_FAIL (4048)	The server encountered an error while executing the request. This value can indicate that the connection failed or was terminated.	No results.

Return value CTBRESULTS returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	A result set is available for processing.
CS_END_RESULTS (-205)	No more result sets are available for processing.
CS_FAIL (-2)	The routine failed.
CS_CANCELLED (-202)	Results were cancelled.

Examples

The following code fragment demonstrates how CTBRESULTS is used to describe a result row for a language request. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to process the result                                   */
/*                                                                    */
/*--------------------------------------------------------------------*/
PROCESS_RESULTS: PROC ;

/*-------------------------------------------------------------*/
/* set up the results data                                     */
/*-------------------------------------------------------------*/

        CALL CTBRESUL( CSL_CMD_HANDLE,
                       CSL_RC,
                       RF_TYPE ) ;

/*-------------------------------------------------------------*/
/* determine the outcome of the comand execution               */
/*-------------------------------------------------------------*/

        SELECT( CSL_RC ) ;

          WHEN( CS_SUCCEED )
          DO ;

/*-------------------------------------------------------------*/
/* determine the type of result returned by the current request */
/*-------------------------------------------------------------*/

            SELECT( RF_TYPE ) ;

/*-------------------------------------------------------------*/
/* process row results                                         */
/*-------------------------------------------------------------*/

              WHEN( CS_ROW_RESULT )
              DO ;
```

```
                CALL RESULT_ROW_PROCESSING ;
                DO WHILE( ^NO_MORE_ROWS ) ;
                  CALL FETCH_ROW_PROCESSING ;
                END ;
              END ;

/*-------------------------------------------------------------*/
/* process parameter results --- there should be no parameter */
/* to process                                                  */
/*-------------------------------------------------------------*/

              WHEN( CS_PARAM_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;
              END ;

/*-------------------------------------------------------------*/
/* process status results --- the stored procedure status     */
/* result will not be processed in this example               */
/*-------------------------------------------------------------*/

              WHEN( CS_STATUS_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;
              END ;

/*-------------------------------------------------------------*/
/* print an error message if the server encountered an error  */
/* while executing the request                                */
/*-------------------------------------------------------------*/

              WHEN( CS_CMD_FAIL )
              DO ;
                NO_ERRORS_SW = FALSE ;
                MSGSTR       =
                  'CTBRESUL returned CS_CMD-FAIL restype' ;
                CALL ERROR_OUT ;
              END ;

/*-------------------------------------------------------------*/
/* print a message for successful commands that returned no   */
/* data( optional )                                           */
/*-------------------------------------------------------------*/

              WHEN( CS_CMD_SUCCEED )
              DO ;
```

```
                MSGSTR = 'CTBRESUL returned CS_CMD_SUCCEED restype' ;
              END ;

/*-------------------------------------------------------------*/
/* print a message for requests that have been processed       */
/* successfully( optional )                                    */
/*-------------------------------------------------------------*/

              WHEN( CS_CMD_DONE )
              DO ;
                MSGSTR = 'CTBRESUL returned CS_CMD_DONE restype' ;
              END ;

              OTHERWISE
              DO ;
                NO_MORE_RESULTS = TRUE ;
                NO_ERRORS_SW    = FALSE ;
                MSGSTR          = 'CTBRESUL returned UNKNOWN restype' ;
                CALL ERROR_OUT ;

              END ;
            END ; /* end of SELECT( RF_TYPE ) */
          END ;

 /*-------------------------------------------------------------*/
 /* print an error message if the CTBRESULTS call failed       */
 /*-------------------------------------------------------------*/

          WHEN( CS_FAIL )
          DO ;
            NO_MORE_RESULTS = TRUE ;
            NO_ERRORS_SW    = FALSE ;
            MSGSTR          = 'CTBRESUL returned CS_FAIL ret_code' ;
            CALL ERROR_OUT ;
          END ;

 /*-------------------------------------------------------------*/
 /* drop out of the results loop if no more result sets are    */
 /* available for processing or if the results were cancelled  */
 /*-------------------------------------------------------------*/

          WHEN( CS_END_RESULTS )
          DO ;
            NO_MORE_RESULTS = TRUE ;
          END ;
```

```
            WHEN( CS_CANCELLED )
            DO ;
              NO_MORE_RESULTS = TRUE ;
            END ;

            OTHERWISE
            DO ;
              NO_MORE_RESULTS = TRUE ;
              NO_ERRORS_SW    = FALSE ;
              MSGSTR          =
                 'CTBRESUL returned unknown ret_code' ;
              CALL ERROR_OUT ;
            END ;
          END ; /* end of SELECT( CSL_RC ) */

          RF_TYPE = 0 ;

END PROCESS_RESULTS ;
```

Usage

- CTBRESULTS tells the application what kind of results returned and sets up result data for processing. An application calls CTBRESULTS after sending a request to the server through CTBSEND and before binding and retrieving the results of that request (if any) with CTBBIND and CTBFETCH.
- “Result data” is an umbrella term for all the types of data that a server can return to an application:
  - Regular rows
  - Return parameters
  - Stored procedure return status

  CTBRESULTS is used to set up all of these types of results for processing.
- Result data is returned to an application in the form of result sets. A result set includes only a single type of result data. For example, a regular row result set contains only regular rows, and a return parameter result set contains only return parameters.

The *CTBRESULTS* loop

- Because a request can generate multiple result sets, an application must call CTBRESULTS as long as it continues to return CS_SUCCEED, indicating that results are available. The simplest way to do this is in a loop that terminates when CTBRESULTS fails to return CS_SUCCEED. After the loop, an application can test the CTBRESULTS final return code to determine why the loop terminated.
- Results are returned to an application in the order in which they are produced. However, this order is not always easy to predict. For example, when an application calls a stored procedure or transaction that in turn calls another stored procedure or transaction, the application might receive a number of row result sets, as well as a return parameter and a return status result set. The order in which these results are returned depends on how the called stored procedure or transaction is written.

  For this reason, we recommend that you include a series of IF statements in your application, ending with a statement that handles all types of results that can be received. The RESULT_TYP argument indicates what type of result data the result set contains.

When are the results of a command completely processed?

- CTBRESULTS sets the result type to CS_CMD_DONE to indicate that the results of a logical command processed completely. A logical command is any command defined through CTBCOMMAND, with the following rules:
  - Each Transact-SQL select statement inside a stored procedure is a logical command. Other Transact-SQL statements inside stored procedures do not count as logical commands.
  - Each Transact-SQL statement in a language request is a logical command.
- A result type of CS_CMD_SUCCEED or CS_CMD_FAIL is immediately followed by a result type of CS_CMD_DONE.

Canceling results

- To cancel remaining results from a request (and eliminate the need to continue calling CTBRESULTS until it fails to return CS_SUCCEED), call CTBCANCEL.

CTBRESULTS and stored procedures

- A runtime error on a language request containing an execute statement returns CS_CMD_FAIL. However, a run-time error on a statement inside a stored procedure or transaction does not return CS_CMD_FAIL. For example, if a called stored procedure or transaction contains an insert statement and the user does not have insert permission on the database table, the insert statement fails, but CTBRESULTS still returns CS_SUCCEED.

  To check for runtime errors inside stored procedures or transactions, examine the procedure return status. This value is returned as a return status result set immediately following the row and parameter results, if any. If the error generates a server message, the message is also available to the application.

  If results are coming from Open ServerConnect, a return status of TDS_DONE_ERROR indicates an error.

See also

Related functions:

- CTBBIND on page 59
- CTBCOMMAND on page 84
- CTBDESCRIBE on page 112
- CTBFETCH on page 139
- CTBSEND on page 174

Related topics:

- “Remote procedure calls (RPCs)” on page 44
- “Results” on page 47

CTBSEND

Description

Sends a request to the server.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 COMMAND      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);

 CALL CTBSEND (COMMAND, RETCODE);
```

Parameters

COMMAND

(I) Handle for this client/server operation. This handle is defined in the associated CTBCMDALLOC call.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

Return value

CTBSEND returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. This result can indicate that SNA sessions will not come up.
CS_CANCELLED (-202)	The routine was cancelled. **Note** This value is returned by SNA sessions only, and is never returned when sending a request to another CICS region.

Examples

Example 1

This code fragment demonstrates the use of CTBSEND. It is taken from the sample program SYCTSAR4 in Appendix B, "Sample RPC Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to allocate, send, and process commands                */
/*                                                                   */
/*-------------------------------------------------------------------*/
SEND_PARAM: PROC ;

/*-----------------------------------------------------------*/
/* allocate a command handle                                 */
/*-----------------------------------------------------------*/

         CALL CTBCMDAL( CSL_CON_HANDLE,
                        CSL_RC,
```

```
                        CSL_CMD_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCMDALLOC failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* prepare the command (an RPC request)                        */
/*-------------------------------------------------------------*/

        PF_STRLEN = STG(CF_CMD);

        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_RPC_CMD,
                       CF_CMD,
                       PF_STRLEN,
                       CS_UNUSED );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCOMMAND failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/*                                                             */
/* setup a return parameter for NUM_OF_ROWS                    */
/*                                                             */
/* describe the first parameter (NUM_OF_ROWS)                  */
/*                                                             */
/*-------------------------------------------------------------*/

        DF_NAME         = '@parm1';
        DF_NAMELEN      = 6;
        DF_DATATYPE     = CS_INT_TYPE;
        DF_FORMAT       = CS_FMT_UNUSED;
        DF_MAXLENGTH    = CS_UNUSED;
        DF_STATUS       = CS_RETURN;
        DF_USERTYPE     = CS_UNUSED;
```

```
        PM_LEN          = STG(PM_PARAM1);
        PM_PARAM1       = 0;                    /* NUM_OF_ROWS */
        PM_NULLIND      = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
                       CSL_RC,
                       DATAFMT,
                       PM_PARAM1,
                       PM_LEN,
                       PM_NULLIND );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBPARAM CS_INT_TYPE parm1 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-----------------------------------------------------------------*/
/*                                                                 */
/* describe the second parameter (DEPTNO)                          */
/*                                                                 */
/*-----------------------------------------------------------------*/

        DF_NAME         = '@parm2';
        DF_NAMELEN      = 6;
        DF_DATATYPE     = CS_VARCHAR_TYPE;
        DF_FORMAT       = CS_FMT_UNUSED;
        DF_MAXLENGTH    = CS_UNUSED;
        DF_STATUS       = CS_INPUTVALUE;
        DF_USERTYPE     = CS_UNUSED;

        PM_PARAM2       = PF_DEPT;                  /* DEPTNO */
        PM_LEN          = PF_DEPT_SIZE ;
        PM_NULLIND      = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
                       CSL_RC,
                       DATAFMT,
                       PM_PARAM2,
                       PM_LEN,
                       PM_NULLIND );
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBPARAM CS_VARCHAR_TYPE parm2 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* send the command                                            */
/*-------------------------------------------------------------*/

        CALL CTBSEND( CSL_CMD_HANDLE,
                      CSL_RC ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBSEND failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

END SEND_PARAM ;
```

Example 2

The following code fragment demonstrates the use of CTBSEND. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to allocate, send, and process commands                */
/*                                                                   */
/*-------------------------------------------------------------------*/
SEND_COMMAND: PROC ;

/*-------------------------------------------------------------*/
/* find out what the maximum number of connections is          */
/*-------------------------------------------------------------*/

        CALL CTBCONFI( CSL_CTX_HANDLE,
                       CSL_RC,
                       CS_GET,
                       CS_MAX_CONNECT,
                       CF_MAXCONNECT,
```

```
                        STG(CF_MAXCONNECT),
                        CS_FALSE,
                        CF_OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCONFIG failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* display number of connections                               */
/*-------------------------------------------------------------*/

        OR2_MAXCONNECT = CF_MAXCONNECT;

/*-------------------------------------------------------------*/
/* allocate a command handle                                   */
/*-------------------------------------------------------------*/

        CALL CTBCMDAL( CSL_CON_HANDLE,
                       CSL_RC,
                       CSL_CMD_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCMDALLOC failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* prepare the language request                                */
/*-------------------------------------------------------------*/

        PF_STRLEN = STG(CF_LANG2 ) ;

        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_LANG_CMD,
                       CF_LANG2,
                       PF_STRLEN,
                       CS_UNUSED ) ;
```

```
            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBCOMMAND failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE ;
            END ;

    /*-------------------------------------------------------------*/
    /* send the language request                                   */
    /*-------------------------------------------------------------*/

            CALL CTBSEND( CSL_CMD_HANDLE,
                          CSL_RC ) ;

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBSEND failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE ;
            END ;

    END SEND_COMMAND ;
```

Usage

- CTBSEND signals the end of the data to be sent to a server (no more parameters, data, messages) and sends a request to the server.
- Sending a request to a server is a three-step process. To send a request to a server, an application:
  - Initiates the request by calling CTBCOMMAND, which initiates a language request, RPC, or message stream to send to the server.
  - Describes parameters for the request, using CTBPARAM.

    Not all requests require parameters. For example, a remote procedure call may or may not require parameters, depending on the stored procedure or transaction being called.
  - Calls CTBSEND to send the request stream to the server.
- CTBSEND does not wait for a response from the server. An application must call CTBRESULTS to verify the success of the request and to set up the results for processing.

See also

Related functions:

- CTBCOMMAND on page 84
- CTBFETCH on page 139
- CTBPARAM on page 149
- CTBRESULTS on page 167

CSBCONFIG

Description

Sets or retrieves context structure properties.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONTEXT        FIXED BIN(31) INIT(0);
   01 RETCODE        FIXED BIN(31) INIT(0);
   01 ACTION         FIXED BIN(31);
   01 PROPERTY       FIXED BIN(31);
   01 BUFFER type;
   01 BUFFER_LEN     FIXED BIN(31);
   01 BUFBLANKSTRIP FIXED BIN(31);
   01 OUTLEN         FIXED BIN(31);

CALL CSBCONFIG (CONTEXT, RETCODE, ACTION, PROPERTY,
BUFFER, BUFFER_LEN, BUFBLANKSTRIP, OUTLEN);
```

Parameters

CONTEXT

(I) A context structure for which the properties are being set or retrieved. The context structure is defined in the program call CSBCTXALLOC.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

ACTION

(I) Action to be taken by this call. *ACTION* is an integer variable that indicates the purpose of this call. Assign *ACTION* one of the following symbolic values:

Value	Meaning
CS_GET (33)	Retrieves the value of the property.
CS_SET (34)	Sets the value of the property.
CS_CLEAR (35)	Clears the value of the property by resetting the property to its default value.

PROPERTY

(I) Symbolic name of the property for which the value is being set or retrieved. Table 3-19 lists the properties that can be set or retrieved by CSBCONFIG.

Table 3-19: Values for property (CSBCONFIG)

Application action	Property	Indicates
Set, retrieve, or clear	CS_EXTRA_INF	Whether to return the extra information required when processing messages in line, using the SQLCA or SQLCODE structures.
Retrieve only	CS_VERSION	The version number of Open Client currently in use.

BUFFER

(I/O) Variable (buffer) that contains the specified property value.

If *ACTION* is CS_SET, CSBCONFIG takes the value from this buffer.

If *ACTION* is CS_GET, CSBCONFIG returns the requested information to this buffer.

If *ACTION* is CS_CLEAR, this value is zeroes.

This argument is typically one of the following datatypes:

```
01 BUFFERFIXED BIN(n);
01 BUFFER  CHAR(n);
```

BUFFER_LEN

(I) Length, in bytes, of the buffer.

If *ACTION* is CS_SET and the value in the buffer is a fixed-length or symbolic value, *BUFFER_LEN* should have a value of CS_ UNUSED. To indicate that the terminating character is the last non-blank character, an application sets *BUFBLANKSTRIP* to CS_TRUE.

If *ACTION* is CS_GET and *BUFFER* is too small to hold the requested information, CSBCONFIG sets *OUTLEN* to the length of the requested information and returns CS_FAIL. To retrieve all the requested information, change the value of *BUFFER_LEN* to the length returned in *OUTLEN* and rerun the application.

If *ACTION* is CS_CLEAR, this value is zeroes.

BUFBLANKSTRIP

(I) Blank stripping indicator. Indicates whether trailing blanks are stripped. Assign this argument one of the following symbolic values:

Value	Meaning
CS_TRUE (1)	Trailing blanks are stripped. The value in the buffer ends at the last non-blank character.
CS_FALSE (0)	Trailing blanks are not stripped. They are included in the value.

OUTLEN

(O) Length, in bytes, of the retrieved information. *OUTLEN* is an integer variable where CSBCONFIG returns the length of the property value being retrieved.

When the retrieved information is larger than *BUFFER_LEN* bytes, an application uses the value of *OUTLEN* to determine how many bytes are needed to hold the information.

OUTLEN is used only when *ACTION* is CS_GET. When the *ACTION* is CS_SET or CS_CLEAR, this value is zeroes.

Return value

CSBCONFIG returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_INVALID_PARAMETER (-4)	One of the CSBCONFIG arguments contains an illegal value. The most likely cause for this code is that a property value is being set and the value assigned to BUFBLANKSTRIP is not CS_TRUE.

Usage

Note CSBCONFIG and CTBCONFIG both set and retrieve context properties. CSBCONFIG is used with global context properties; CTBCONFIG is used with Client-Library properties.

- CSBCONFIG can be used to set and retrieve the values of CS_EXTRA_INF and to retrieve the version number of Open Client currently in use.
- Use CTBCONFIG to set and retrieve the values of Client-Library-specific context properties. Properties set through CTBCONFIG affect only Client-Library behaviors.

About global context properties

- CS_EXTRA_INF determines whether or not Client-Library returns the extra information that is required to fill in a SQLCA or SQLCODE structure.
- If an application is not retrieving messages into a SQLCA or SQLCODE structure, the extra information is returned as ordinary Client-Library messages.

Version level

- The CS_VERSION property represents the version of Client-Library behavior that an application requests through CSBCTXALLOC.
- An application can only retrieve the value of CS_VERSION; it cannot assign a value to CS_VERSION.

User data

- The CS_USERDATA property defines user-allocated data. This property allows an application to associate user data with a particular connection or command structure. An application allocates a data space from which it can get this data when needed.
- A PL/I program can use the WORK AREAS section to define this data.

See also

Related functions:

- CSBCTXALLOC on page 190
- CTBCONPROPS on page 105
- CTBCONFIG on page 98
- CTBINIT on page 147

CSBCONVERT

Description

Converts a data value from one datatype to another.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
    01 CONTEXT        FIXED BIN(31) INIT(0);
    01 RETCODE        FIXED BIN(31); INIT(0);
    01 SRCFMT,
            05 SRC_NAME          CHAR(132),
            05 SRC_NAMELEN       FIXED BIN(31),
            05 SRC_TYPE          FIXED BIN(31),
```

```
                05 SRC_FORMAT        FIXED BIN(31),
                05 SRC_MAXLEN        FIXED BIN(31),
                05 SRC_SCALE         FIXED BIN(31),
                05 SRC_PRECIS        FIXED BIN(31),
                05 SRC_STATUS        FIXED BIN(31),
                05 SRC_COUNT         FIXED BIN(31),
                05 SRC_UTYPE         FIXED BIN(31),
                05 SRC_LOCALE        FIXED BIN(31);
    01 SRCDATA      type;
    01 DESTFMT,
                05 DEST_NAME         CHAR(132),
                05 DEST_NAMELEN      FIXED BIN(31),
                05 DEST_TYPE         FIXED BIN(31),
                05 DEST_FORMAT       FIXED BIN(31),
                05 DEST_MAXLEN       FIXED BIN(31),
                05 DEST_SCALE        FIXED BIN(31),
                05 DEST_PRECIS       FIXED BIN(31),
                05 DEST_STATUS       FIXED BIN(31),
                05 DEST_COUNT        FIXED BIN(31),
                05 DEST_UTYPE        FIXED BIN(31),
                05 DEST_LOCALE       FIXED BIN(31);
    01 DESTDATA     type;
    01 OUTLEN       FIXED BIN(31);

    CALL CSBCONVE (CONTEXT, RETCODE, SRCFMT, SRCDATA,
     DESTFMT, DESTDATA, OUTLEN);
```

Parameters

CONTEXT

(I) A context structure. The context structure is defined in the program call CSBCTXALLOC.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under “Returns,” in this section.

SRCFMT

(I) A structure that describes the variable(s) that contain the source data. CSBCONVERT ignores SRCFMT fields that it does not use.

Table 3-20 lists the fields in the SRCFMT structure and indicates whether and how they are used by CSBCONVERT. For a general discussion of this structure, see “DATAFMT structure” on page 26.

Table 3-20: Fields in the SRCFMT structure for CSBCONVERT

Field	When used	Value represents
SRC_NAME	Not used (CS_FMT_UNUSED).	Not applicable.
SRC_NAMELEN	Not used (CS_FMT_UNUSED).	Not applicable.

Field	When used	Value represents
SRC_TYPE	For all datatype conversions.	The datatype of the source data. CSBCONVERT converts this datatype to the datatype specified for the destination variable (DEST_TYPE).
SRC_FORMAT	Not used (CS_FMT_UNUSED).	Not applicable.
SRC_MAXLEN	When converting non-fixed-length source datatypes to any destination type. SRC_MAXLEN is ignored when converting fixed-length types.	The length of the source variable, in bytes. If SRCDATA is an array, SRC_MAXLEN is the length of an element in the array. When converting character or binary datatypes, SRC_MAXLEN must describe the total length of the source variable, including any space required for special terminating bytes, with this exception: when converting a VARYCHAR-type source such as the DB2 VARCHAR, SRC_MAXLEN does not include the length of the "LL" length specification. In case of Sybase-numeric, Sybase-decimal or packed decimal, this value is the actual length.
SRC_SCALE	Only when converting to or from numeric, Sybase-decimal, or packed decimal datatypes.	Number of digits that follow the decimal point in the source data. SRC_SCALE must be less than or equal to SRC_PRECIS and cannot be greater than 31.
SRC_PRECIS	Only when converting to or from packed decimal, numeric and Sybase-decimal datatypes.	The total number of digits in the source data. SRC_PRECIS must be greater than or equal to SRC_SCALE and cannot be less than 1 or greater than 31.
SRC_STATUS	Not used (CS_FMT_UNUSED).	Not applicable.
SRC_COUNT	Not used (CS_FMT_UNUSED).	Not applicable.
SRC_UTYPE	Not used (CS_FMT_UNUSED).	Not applicable.
SRC_LOCALE	Not used (CS_FMT_UNUSED).	Not applicable.

SRCDATA

(I) Name of the source variable that contains the data to be converted. This is the variable described in the SRCFMT structure.

DESTFMT

(I) A structure that contains a description of the variable(s) that contain destination (converted) data. CSBCONVERT ignores DESTFMT fields that it does not use.

Table 3-21 lists the fields in the *DESTFMT* structure and indicates whether and how they are used by CSBCONVERT. For a general discussion of this structure, see "DATAFMT structure" on page 26.

Table 3-21: Fields in the DATAFMT structure for CSBCONVERT

Field	When used	Value represents
DEST_NAME	Not used (CS_FMT_UNUSED).	Not applicable.
DEST_NAMELEN	Not used (CS_FMT_UNUSED).	Not applicable.
DEST_TYPE	For all datatype conversions.	The datatype of the destination variable. CSBCONVERT converts the datatype specified for the source data (SRCTYPE) to this datatype.
DEST_FORMAT	Not used (CS_FMT_UNUSED).	Not applicable.
DEST_MAXLEN	When converting all source datatypes to non-fixed-length datatypes. DEST_MAXLEN is ignored when converting to fixed-length datatypes.	The length of the destination variable, in bytes. If DESTDATA is an array, DEST_MAXLEN is the length of an element in the array. When converting character or binary datatypes, DEST_MAXLEN must describe the total length of the destination variable, including any space required for special terminating bytes, with this exception: when converting to a VARYCHAR-type destination such as the DB2 VARCHAR, DEST_MAXLEN does not include the length of the "LL" length specification. DEST_MAXLEN = 35 when converting to numeric or Sybase-decimal.
DEST_SCALE	Only when converting to or from numeric, Sybase-decimal or packed decimal datatypes.	Number of digits that follow the decimal point in the destination variable. DEST_SCALE must be less than or equal to DEST_PRECIS and cannot be greater than 31. Use the same value as in SRC_SCALE.
DEST_PRECIS	Only when converting to or from numeric, Sybase-decimal or packed decimal datatypes.	The total number of digits in the destination data. DEST_PRECIS must be greater than or equal to DEST_SCALE and cannot be less than 1 or greater than 31. Use the same value as in SRC_PRECIS.
DEST_STATUS	Not used (CS_FMT_UNUSED).	Not applicable.

Field	When used	Value represents
DEST_COUNT	Not used (CS_FMT_UNUSED).	Not applicable.
DEST_UTYPE	Not used (CS_FMT_UNUSED).	Not applicable.
DEST_LOCALE	Not used (CS_FMT_UNUSED).	Not applicable.

DESTDATA

(O) Name of the variable that contains the converted data. This is the variable described in the DESTDATA structure.

OUTLEN

(O) Actual length, in bytes, of the data placed in DESTDATA. If the conversion fails, CSBCONVERT sets OUTLEN to CS_UNUSED.

Return value

CSBCONVERT returns one of the following value listed in Table 3-22.

Table 3-22: CSBCONVERT return values

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed.
TDS_INVALID_DATAFMT_VALUE (-181)	A SRCFMT or DESTFMT field contains an illegal value—probably an illegal datatype value.
TDS_INVALID_VAR_ADDRESS (-175)	This value cannot be NULL.
TDS_MONEY_CONVERSION_ERROR (-22)	Converting TDSMONEY4 failed, possibly because the TDS version is not 4.2 or above.
TDS_INVALID_DATA_CONVERSION (-172)	This value cannot be NULL.
TDS_INVALID_LENGTH (-173)	Converting TDSMONEY4 failed, possibly because the TDS version is not 4.2 or above.

Examples

The following code fragment illustrates the use of CSBCONVERT to convert a column in result rows. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to fetch row processing                                 */
/*                                                                    */
/*--------------------------------------------------------------------*/
FETCH_ROW_PROCESSING: PROC ;

        CALL CTBFETCH( CSL_CMD_HANDLE,
                       CSL_RC,
```

```
                    CS_UNUSED,       /* type   */
                    CS_UNUSED,       /* offset */
                    CS_UNUSED,       /* option */
                    FF_ROWS_READ ) ;

SELECT( CSL_RC ) ;

  WHEN( CS_SUCCEED )
  DO ;
    NO_MORE_ROWS          = FALSE ;
    CF_COL_FIRSTNME_CHAR = BLANK ;
    DF_DATATYPE           = CS_VARCHAR_TYPE;
    DF_MAXLENGTH          = LENGTH( CF_COL_FIRSTNME ) ;
    DF2_DATATYPE          = CS_CHAR_TYPE;
    DF2_MAXLENGTH         = STG(CF_COL_FIRSTNME_CHAR);

    CALL CSBCONVE( CSL_CTX_HANDLE,
                   CSL_RC,
                   DATAFMT,
                   CF_COL_FIRSTNME,
                   DATAFMT2,
                   CF_COL_FIRSTNME_CHAR,
                   CF_COL_LEN);

    IF CSL_RC ^= CS_SUCCEED THEN
    DO ;
      MSGSTR       = 'CSCONVERT CS_CHAR_TYPE failed' ;
      NO_ERRORS_SW = FALSE ;
      CALL ERROR_OUT;
      CALL ALL_DONE;
    END ;

    FF_ROW_NUM = FF_ROW_NUM + 1 ;
```

Usage

- A client application can use this function to convert the datatype of RPC return parameters to the datatype of the target server, and to convert the datatype of a retrieved value to a datatype that can be used by Open ClientConnect. This function converts a single variable each time it executes.
- When converting columns, an application must issue a separate CSBCONVERT call for each column to be converted. If several rows of data need converting, the application must issue a separate CSBCONVERT call for every column that needs conversion in each row.
- Table 3-23 lists the conversions you can perform with CSBCONVERT.

Table 3-23: Conversions performed by CSBCONVERT

Source type	Result type	Notes
CS_VARCHAR	CS_CHAR	Does EBCDIC to ASCII conversion; pads with blanks.
CS_CHAR	CS_VARCHAR	
CS_MONEY	CS_CHAR	
CS_MONEY	CS_VARCHAR	
CS_REAL	CS_FLOAT	Truncates low order digits.
CS_MONEY	CS_FLOAT	
CS_PACKED370	CS_FLOAT	
CS_FLOAT	CS_REAL	Pads with zeroes.
CS_MONEY	CS_PACKED370	
CS_CHAR	CS_PACKED370	
CS_VARCHAR	CS_PACKED370	
CS_FLOAT	CS_PACKED370	
CS_NUMERIC	CS_PACKED370	
CS_DECIMAL	CS_PACKED370	
CS_NUMERIC	CS_CHAR	
CS_DECIMAL	CS_CHAR	
CS_DATETIME	CS_CHAR	
CS_CHAR	CS_NUMERIC	
CS_CHAR	CS_DECIMAL	
CS_PACKED370	CS_NUMERIC	
CS_PACKED370	CS_DECIMAL	
CS_PACKED370	CS_CHAR	

Warning! Converting CS_MONEY or CS_CHAR values to CS_FLOAT can result in a loss of precision. Converting a CS_FLOAT value to a character type can also result in a loss of precision.

See also

Related functions:

- CTBFETCH on page 139

Related topics:

- “Datatypes” on page 30
- “DATAFMT structure” on page 26

CSBCTXALLOC

Description Allocates a context structure.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 VERSION      FIXED BIN(31);
   01 RETCODE      FIXED BIN(31) INIT(0);
   01 CONTEXT      FIXED BIN(31) INIT(0);

 CALL CSBCTXAL (VERSION, RETCODE, CONTEXT);
```

Parameters

VERSION

(I) Version of Client-Library behavior that the application expects. The following table lists the symbolic values that are legal for *VERSION*:

Value	Indicates	Supported features
CS_VERSION_46	Communicates with Adaptive Server release 4.6.	RPCs. **Note** This is the initial version of Client-Library.
CS_VERSION_50	Communicates with Adaptive Server release 10.0 and above.	RPCs.

RETCODE

(O) Variable where the result from an executed function returns. Its value is one of the codes listed under "Returns," in this section.

CONTEXT

(O) Variable where this newly-allocated context structure returns. This is the name used by CTBINIT when it initializes Client-Library.

Return value CSBCTXALLOC returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. The most common cause for a CSBCTXALLOC failure is a lack of available memory.

Examples The following code fragment demonstrates how CSBCTXALLOC works with other functions to initialize a program. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*--------------------------------------------------------------------*/
/* program initialization                                             */
/*--------------------------------------------------------------------*/
```

```
        DIAG_MSGS_INITIALIZED = TRUE ;
        MSG_TEXT_2            = 'Press Clear To Exit';
        NO_ERRORS_SW          = TRUE ;
        PAGE_CNT              = PAGE_CNT + 1;
        SERVERL               = -1 ;

        DO I1 = 1 TO 13 ;
          RSLTNO( I1 ) = BLANK ;
        END ;

        CALL GET_SYSTEM_TIME ;

GET_INPUT_AGAIN:

          CALL DISPLAY_INITIAL_SCREEN ;
          CALL GET_INPUT_DATA ;

/*-------------------------------------------------------------------*/
/* allocate a context structure                                      */
/*-------------------------------------------------------------------*/

        CALL CSBCTXAL( CS_VERSION_50,
                       CSL_RC,
                       CSL_CTX_HANDLE );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CSCTXALLOC failed';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;

/*-------------------------------------------------------------------*/
/* initialize the Client-Library                                     */
/*-------------------------------------------------------------------*/

        CALL CTBINIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_VERSION_50 );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CTBINIT failed';
            NO_ERRORS_SW = FALSE ;
```

```
        CALL ERROR_OUT;
        CALL ALL_DONE;
END;

CALL PROCESS_INPUT ;

CALL QUIT_CLIENT_LIBRARY ;
```

Usage

- CSBCTXALLOC allocates a context structure.
- A context structure contains information that describes an application context. For example, a context structure defines the version of Client-Library that is in use.
- Allocating a context structure is the first step in any Client-Library application.
- After allocating a context structure, a Client-Library application typically customizes the context by calling CSBCONFIG and/or CTBCONFIG, then sets up one or more connections within the context.
- To deallocate a context structure, an application calls CSBCTXDROP.

See also

Related functions:

- CSBCONFIG on page 180
- CTBCONALLOC on page 88
- CTBCONFIG on page 98
- CSBCTXDROP on page 192

CSBCTXDROP

Description

Deallocates a context structure.

Syntax

```
%INCLUDE CTPUBLIC;

DCL
   01 CONTEXT      FIXED BIN(31) INIT(0);
   01 RETCODE      FIXED BIN(31) INIT(0);

CALL CSBCTXDR (CONTEXT, RETCODE);
```

Parameters

CONTEXT

(I) A context structure.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed under "Returns," in this section.

Return value

CSBCTXDROP returns one of the following values:

Value	Meaning
CS_SUCCEED (-1)	The routine completed successfully.
CS_FAIL (-2)	The routine failed. The most common cause for a CSBCTXDROP failure is that the context contains an open connection.

Examples

The following code fragment demonstrates how CSBCTXDROP is used with other functions at the end of a program to close the connection and return to CICS. It is taken from the sample program SYCTSAA4 in Appendix A, "Sample Language Application."

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to perform exit client library and deallocate context */
/* structure.                                                        */
/*                                                                   */
/*-------------------------------------------------------------------*/
 QUIT_CLIENT_LIBRARY: PROC ;

 /*-------------------------------------------------------------------*/
 /* exit the Client Library                                           */
 /*-------------------------------------------------------------------*/

         CALL CTBEXIT( CSL_CTX_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;

         IF CSL_RC = CS_FAIL THEN
         DO ;
           MSGSTR = 'CTBEXIT failed' ;
           CALL ERROR_OUT ;
         END ;
/*-------------------------------------------------------------------*/
/* de-allocate the context structure                                 */
/*-------------------------------------------------------------------*/

         CALL CSBCTXDR( CSL_CTX_HANDLE,
                        CSL_RC ) ;
```

```
IF CSL_RC = CS_FAIL THEN
          DO ;
            MSGSTR = 'CSBCTXDROP failed' ;
            CALL ERROR_OUT ;
          END ;
EXEC CICS RETURN ;
END QUIT_CLIENT_LIBRARY ;
```

Usage

- CSBCTXDROP deallocates a context structure.
- A context structure describes a particular context, or operating environment, for a set of server connections.
- Once a context has been deallocated, it cannot be reused. To allocate a new context, an application calls CSBCTXALLOC.
- A Client-Library application cannot call CSBCTXDROP to deallocate a context structure until it has called CTBEXIT to clean up Client-Library space associated with the context.
- CSBCTXDROP fails if the context contains an open connection.

See also

Related functions:

- CSBCTXALLOC on page 190
- CTBCLOSE on page 72
- CTBEXIT on page 136

APPENDIX A

Sample Language Application

This appendix contains a sample Open ClientConnect program that sends a language request to an Adaptive Server Enterprise. It retrieves information from a table, SYBASE.SAMPLETB, on the target server.

The purpose of this sample program is to demonstrate the use of Client-Library functions, particularly those designed to send language requests. In some cases, one Client-Library function is used for demonstration purposes when another function would be more efficient. In order to best illustrate the flow of processing, the program does not do extensive error checking.

This sample program is provided as part of the Open ClientConnect package, on the API. When the target sever is an Adaptive Server Enterprise, the Transaction Router Service (TRS) administrator can create the table SYBASE.SAMPLETB on that server with a script provided with TRS.

An additional sample program, SYCTSAL4, is provided as part of the Open ClientConnect package on the API. (It is not documented in this guide.) The SYCTSAL4 program demonstrates how to send a language request with parameters to an Adaptive Server Enterprise or Open ServerConnect running in a CICS/IMS region. The table SYBASE.SAMPLETB is provided on the Open ServerConnect API.

Sample program – SYCTSAA4

```
SYCTSAA4: PROC OPTIONS(MAIN REENTRANT);
/*        @(#) syctsaa4.pli 11.3 12/14/95                                  */
/******        SYCTSAA4 - CLIENT LANGUAGE REQUEST APPL - PL/I - CICS ********/
/*            CICS TRANID:  SYA4 /* /*  PROGRAM:     SYCTSAA4
/*           PURPOSE:  Demonstrates    Open Client for CICS CALLs.
*/
/*
/*           FUNCTION: Illustrates how to send a language request with
*/
```

```
/*            parameters to: /*          - A SQL Server
/*                                                                  */
/*                                                                  */
/*            SQL Server:                                          */
/*                                                                  */
/*              If the request is sent to a SQL Server it          */
/*              executes the SQL statement:                        */
/*                                                                  */
/*                  SELECT  FIRSTNME, EDUCLVL                      */
/*                    FROM  SYBASE.SAMPLETB                        */
/*                                                                  */
/*              Note: The Net-Gateway/MCG product includes a script */
/*                    that creates this procedure in a target SQL   */
/*                    server.                                       */
/*                                                                  */
/*                                                                  */
/*  PREREQS:  Before running SYCTSAA4, make sure that the server    */
/*            you wish to access has an entry in the Connection     */
/*            Router Table for that Server and the MCG(s) that      */
/*            you wish to use.                                      */
/*                                                                  */
/* INPUT:    On the input screen, make sure to enter the Server     */
/*           name, user id, and password for the target server.     */
/*           TRAN NAME is not used for LAN servers.                 */
/*                                                                  */
/*                                                                  */
/*  Open Client CALLs used in this sample:                          */
/*                                                                  */
/*    CSBCONVERT    convert a datatype from one value to another    */
/*    CSBCTXALLOC   allocate a context                              */
/*    CSBCTXDROP    drop a context                                  */
/*    CTBBIND       bind a column variable                          */
/*    CTBCLOSE      close a server connection                       */
/*    CTBCONFIG     set or retrieve context properties              */
/*    CTBCMDALLOC   allocate a command                              */
/*    CTBCMDDROP    drop a command                                  */
/*    CTBCOMMAND    initiate remote procedure call                  */
/*    CTBCONALLOC   allocate a connection                           */
/*    CTBCONDROP    drop a connection                               */
/*    CTBCONPROPS   alter properties of a connection                */
/*    CTBCONNECT    open a server connection                        */
/*    CTBDESCRIBE   return a description of result data             */
/*    CTBDIAG       retrieve SQLCODE messages                       */
/*    CTBEXIT       exit client library                             */
/*    CTBFETCH      fetch result data                               */
/*    CTBINIT       init client library                             */
```

```
/*    CTBRESULTS    set up result data                                  */
 /*    CTBRESINFO    return result set info                              */
 /*    CTBSEND       send a request to the server                        */
 /*                                                                      */
 /* History:                                                             */
 /*                                                                      */
 /* Date    BTS#   Descrition                                            */
 /* ======= ====== ============================================== */
 /* Feb1795        Create                                                */
 /* Oct3095 99999  Rewrite and add front end to the program              */
 /*                                                                      */
 /*                                                                      */
 /*                                                                      */
 /************************************************************************/

 DCL PLIXOPT CHAR(50) VAR INIT('NOSPIE, NOSTAE') STATIC EXTERNAL;

 /*----------------------------------------------------------------------*/
 /* CLIENT LIBRARY PL/I COPY BOOK                                        */
 /*----------------------------------------------------------------------*/

         %INCLUDE CTPUBLIC;

 /*----------------------------------------------------------------------*/
 /* CICS BMS DEFINITIONS PL/I COPY BOOK                                  */
 /*----------------------------------------------------------------------*/

         %INCLUDE SYCTBA4;

 /*----------------------------------------------------------------------*/
 /* Standard CICS Attribute and Print Control Chararcter List            */
 /*----------------------------------------------------------------------*/

         %INCLUDE DFHBMSCA;

 /*----------------------------------------------------------------------*/
 /* CICS Standard Attention Identifiers PL/I Copy Book                   */
 /*----------------------------------------------------------------------*/

         %INCLUDE DFHAID;
/*----------------------------------------------------------------------*/
/* CLIENT LIB ROUTINES DECLARATIONS                                     */
/*----------------------------------------------------------------------*/
         DCL
             CSBCONVE     ENTRY OPTIONS(INTER ASSEMBLER),
             CSBCTXAL     ENTRY OPTIONS(INTER ASSEMBLER),
```

```
            CSBCTXDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBBIND      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCLOSE     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONFI     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCMDAL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCMDDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCOMMA     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONAL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONPR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONNE     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBDESCR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBDIAG      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBEXIT      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBFETCH     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBINIT      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBRESUL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBRESIN     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBSEND      ENTRY OPTIONS(INTER ASSEMBLER);

/*-------------------------------------------------------------------*/
/* BUILT IN FUNCTIONS DECLARATIONS                                   */
/*-------------------------------------------------------------------*/
        DCL
            ADDR         BUILTIN,
            CSTG         BUILTIN,
            INDEX        BUILTIN,
            LENGTH       BUILTIN,
            STG          BUILTIN,
            SUBSTR       BUILTIN;

        DCL
            SYSPRINT STREAM OUTPUT PRINT ;

/*-------------------------------------------------------------------*/
/* WORK AREAS                                                        */
/*-------------------------------------------------------------------*/
        DCL
            01  INTERNAL_FIELDS,
                05  PARM_CNT              FIXED BIN(31),
                05  NETDRIVER             FIXED BIN(31) INIT(9999);

        DCL
            01  CS_LIB_MISC_FIELDS,
                05  CSL_CMD_HANDLE        FIXED BIN(31) INIT(0),
                05  CSL_CON_HANDLE        FIXED BIN(31) INIT(0),
```

```
        05  CSL_CTX_HANDLE         FIXED BIN(31) INIT(0),
        05  CSL_NULL               FIXED BIN(31) INIT(0),
        05  CSL_RC                 FIXED BIN(31);

DCL
    01  PROPS_FIELDS,
        05  PF_SERVER              CHAR(30)      INIT(' '),
        05  PF_SERVER_SIZE         FIXED BIN(31) INIT(0),
        05  PF_USER                CHAR(08)      INIT(' '),
        05  PF_USER_SIZE           FIXED BIN(31) INIT(0),
        05  PF_PWD                 CHAR(08)      INIT(' '),
        05  PF_PWD_SIZE            FIXED BIN(31) INIT(0),
        05  PF_TRAN                CHAR(08)      INIT(' '),
        05  PF_TRANL               FIXED BIN(31) INIT(0),
        05  PF_NETDRV              CHAR(08)      INIT(' '),
        05  PF_DRV_SIZE            FIXED BIN(31) INIT(0),
        05  PF_STRLEN              FIXED BIN(31),
        05  PF_MSGLIMIT            FIXED BIN(31);

DCL
    01  DIAG_FIELDS,
        05  DF_STATUS              FIXED BIN(31) INIT(0),
        05  DF_MSGNO               FIXED BIN(31) INIT(1),
        05  DF_NUM_OF_MSGS         FIXED BIN(31) INIT(0);

DCL
    01  CONFIG_FIELDS,
        05  CF_MAXCONNECT          FIXED BIN(31),
        05  CF_OUTLEN              FIXED BIN(31);

DCL
    01  QUERY_FIELDS,
        05  QF_LEN                 FIXED BIN(15) INIT(1),
        05  QF_MAXLEN              FIXED BIN(15) INIT(1),
        05  QF_ANSWER              CHAR(01)      INIT(' ');

DCL
    01  CANCELED_FIELDS,
        05  CICS_RESPONSE          FIXED BIN(31);

DCL
    01  FETCH_FIELDS,
        05  FF_ROWS_READ           FIXED BIN(31),
        05  FF_ROW_NUM             FIXED BIN(31) INIT(0);

DCL
```

```
    01  RESINFO_FIELDS,
        05  RF_NUMDATA             FIXED BIN(31);

DCL
    01  OUTPUT_ROW,
        05  OR_COL_FIRSTNME_CHAR   CHAR(12),
        05  SPACE1                 CHAR(01) INIT(' '),
        05  OR_COL_EDLEVEL         PIC'999';

DCL
    01  OUTPUT_ROW_STR             CHAR(16)
                                   DEFINED OUTPUT_ROW;
DCL
    01  OUTPUT_ROW2,
        05  OR2_MESG               CHAR(37)
            INIT('The maximum number of connections is '),
        05  OR2_MAXCONNECT         PIC'99999',
        05  OR2_PERIOD             CHAR(01)      INIT('.');

DCL
    01  OUTPUT_ROW_STR2            CHAR(43)
                                   DEFINED OUTPUT_ROW2;

DCL
    01  OUTPUT_ROW4,
        05  OR4_MESG               CHAR(25)
            INIT('The number of columns is '),
        05  OR4_NUMDATA            PIC'99999',
        05  OR4_PERIOD             CHAR(01)      INIT('.');

DCL
    01  OUTPUT_ROW_STR4            CHAR(31)
                                   DEFINED OUTPUT_ROW4;

DCL
    01  COLUMN_FIELDS,
        05  CF_COL_FIRSTNME        CHAR(12) VAR,
        05  CF_COL_FIRSTNME_CHAR   CHAR(12),
        05  CF_COL_EDLEVEL         FIXED BIN(15),
        05  CF_COL_LEN             FIXED BIN(31),
        05  CF_COL_NULL            FIXED BIN(31) INIT(0),
        05  CF_COL_NUMBER          FIXED BIN(31) INIT(1),
        05  CF_COL_INDICATOR       FIXED BIN(15) INIT(0);

DCL
    01  LANG_FIELDS                STATIC,
```

```
        05  CF_LANG1               CHAR(19)
            INIT('Wrong SQL statement'),
        05  CF_LANG2               CHAR(45)
        INIT('SELECT FIRSTNME, EDUCLVL FROM SYBASE.SAMPLETB');

DCL
    01  RESULTS_FIELDS,
        05  RF_TYPE                FIXED BIN(31);

DCL
    01  DATAFMT,
        05  DF_NAME                CHAR(132),
        05  DF_NAMELEN             FIXED BIN(31),
        05  DF_DATATYPE            FIXED BIN(31),
        05  DF_FORMAT              FIXED BIN(31),
        05  DF_MAXLENGTH           FIXED BIN(31),
        05  DF_SCALE               FIXED BIN(31),
        05  DF_PRECISION           FIXED BIN(31),
        05  DF_STATUS              FIXED BIN(31),
        05  DF_COUNT               FIXED BIN(31),
        05  DF_USERTYPE            FIXED BIN(31),
        05  DF_LOCALE              CHAR(68);

DCL
    01  DATAFMT2,
        05  DF2_NAME               CHAR(132),
        05  DF2_NAMELEN            FIXED BIN(31),
        05  DF2_DATATYPE           FIXED BIN(31),
        05  DF2_FORMAT             FIXED BIN(31),
        05  DF2_MAXLENGTH          FIXED BIN(31),
        05  DF2_SCALE              FIXED BIN(31),
        05  DF2_PRECISION          FIXED BIN(31),
        05  DF2_STATUS             FIXED BIN(31),
        05  DF2_COUNT              FIXED BIN(31),
        05  DF2_USERTYPE           FIXED BIN(31),
        05  DF2_LOCALE             CHAR(68);

DCL
    01 DISP_MSG,
       05 TEST_CASE                CHAR(09) INIT('SYCTSAA4 '),
       05 MSG,
          10 SAMP_LIT              CHAR(05) INIT('rc = '),
          10 SAMP_RC               PIC'99',
          10 REST_LIT              CHAR(15)
                                     INIT('  Result Type: '),
          10 REST_TYPE             PIC'9999',
```

```
            10 FILLER               CHAR(03) INIT(' '),
            10 MSGSTR               CHAR(40) INIT(' ');

      DCL
          BLANK                     CHAR(01)       INIT(' '),
          BLANK_13                  CHAR(13)       INIT(' '),
          FALSE                     BIT(01)        INIT('0'B),
          I1                        FIXED BIN(15)  INIT(0),
          MAX_SCREEN_ROWS           FIXED BIN(15)  INIT(10 ),
          MSG_TEXT_1                CHAR(79)       INIT(' '),
          MSG_TEXT_2                CHAR(79)       INIT(' '),
          OUTLEN                    FIXED BIN(31)  INIT(0),
          PAGE_CNT                  FIXED BIN(15)  INIT(0),
          STRLEN                    FIXED BIN(31)  INIT(0),
          TMP_TIME                  CHAR(08)       INIT(' '),
          TMP_DATE                  CHAR(08)       INIT(' '),
          TRUE                      BIT(01)        INIT('1'B),
          UTIME                     FIXED DEC(15)  INIT(0);

      DCL
          DIAG_MSGS_INITIALIZED     BIT(1)   INIT('0'B),
          ENTER_DATA_SW             BIT(1)   INIT('0'B),
          NO_ERRORS_SW              BIT(1)   INIT('0'B),
          NO_MORE_RESULTS           BIT(1)   INIT('0'B),
          NO_MORE_ROWS              BIT(1)   INIT('0'B),
          PRINT_ONCE                BIT(1)   INIT('1'B);

/*-------------------------------------------------------------------*/
/* Client Message Structure                                          */
/*-------------------------------------------------------------------*/

      DCL
          01 CLIENT_MSG,
             05 CM_SEVERITY          FIXED BIN(31),
             05 CM_MSGNO             FIXED BIN(31),
             05 CM_TEXT              CHAR(256),
             05 CM_TEXT_LEN          FIXED BIN(31),
             05 CM_OS_MSGNO          FIXED BIN(31),
             05 CM_OS_MSGTXT         CHAR(256),
             05 CM_OS_MSGTEXT_LEN    FIXED BIN(31),
             05 CM_STATUS            FIXED BIN(31);

      DCL
          01 DISP_CLIENT_MSG_1,
             05 CM_SEVERITY_HDR      CHAR(13)
                                      INIT('  Severity:  '),
```

```
            05 CM_SEVERITY_DATA        PIC'ZZZ9',
            05 CM_STATUS_HDR           CHAR(12)
                                         INIT(',  Status:  '),
            05 CM_STATUS_DATA          PIC'ZZZ9' ;

        DCL
            01 DISP_CLIENT_MSG_2,
            05 CM_OC_MSGNO_HDR         CHAR(13)
                                         INIT('  OC MsgNo:  '),
            05 CM_OC_MSGNO_DATA        PIC'ZZZZZZZ9' ;

        DCL
            01 DISP_CLIENT_MSG_3,
            05 CM_OC_MSG_HDR           CHAR(13)
                                         INIT('  OC MsgTx:  '),
            05 CM_OC_MSG_DATA          CHAR(66);

        DCL
            01 DISP_CLIENT_MSG_3A,
            05 CM_OC_MSG_DATA_1        CHAR(66),
            05 CM_OC_MSG_DATA_2        CHAR(66),
            05 CM_OC_MSG_DATA_3        CHAR(66),
            05 CM_OC_MSG_DATA_4        CHAR(58);

        DCL
            01 DISP_CLIENT_MSG_3B,
            05 FILLER                  CHAR(13) INIT(' '),
            05 CM_OC_MSG_DATA_X        CHAR(66);

        DCL
            01 DISP_CLIENT_MSG_4,
            05 CM_OS_MSG_HDR           CHAR(13)
                                         INIT('  OS MsgNo:  '),
            05 CM_OS_MSGNO_DATA        PIC'ZZZZZZZ9' ;

        DCL
            01 DISP_CLIENT_MSG_5,
            05 CM_OS_MSG_HDR           CHAR(13)
                                         INIT('  OS MsgTx:  '),
            05 CM_OS_MSG_DATA          CHAR(66);

/*-------------------------------------------------------------------*/
/* Server Message Structure                                          */
/*-------------------------------------------------------------------*/

        DCL
```

```
      01 SERVER_MSG,
         05 SM_MSGNO                FIXED BIN(31),
         05 SM_STATE                FIXED BIN(31),
         05 SM_SEV                  FIXED BIN(31),
         05 SM_TEXT                 CHAR(256),
         05 SM_TEXT_LEN             FIXED BIN(31),
         05 SM_SVRNAME              CHAR(256),
         05 SM_SVRNAME_LEN          FIXED BIN(31),
         05 SM_PROC                 CHAR(256),
         05 SM_PROC_LEN             FIXED BIN(31),
         05 SM_LINE                 FIXED BIN(31),
         05 SM_STATUS               FIXED BIN(31);

    DCL
      01 DISP_SERVER_MSG_1,
         05 SM_MSG_NO_HDR           CHAR(13)
                                     INIT('  Message#:  '),
         05 SM_MSG_NO_DATA          PIC'ZZZZZZZ9',
         05 SM_SEVERITY_HDR         CHAR(14)
                                     INIT(',  Severity:  '),
         05 SM_SEVERITY_DATA        PIC'ZZZ9',
         05 SM_STATE_HDR            CHAR(14)
                                     INIT(',  State No:  '),
         05 SM_STATE_DATA           PIC'ZZZ9' ;

    DCL
      01 DISP_SERVER_MSG_2,
         05 SM_LINE_NO_HDR          CHAR(13)
                                     INIT('  Line  No:  '),
         05 SM_LINE_NO_DATA         PIC'ZZZ9',
         05 SM_STATUS_HDR           CHAR(14)
                                     INIT(',  Status  :  '),
         05 SM_STATUS_DATA          PIC'ZZZ9' ;

    DCL
      01 DISP_SERVER_MSG_3,
         05 SM_SVRNAME_HDR          CHAR(13)
                                     INIT('  Serv Nam:  '),
         05 SM_SVRNAME_DATA         CHAR(66) ;

    DCL
      01 DISP_SERVER_MSG_4,
         05 SM_PROC_ID_HDR          CHAR(13)
                                     INIT('  Proc  ID:  '),
         05 SM_PROC_ID_DATA         CHAR(66);
```

```
        DCL
            01 DISP_SERVER_MSG_5,
               05 SM_MSG_HDR              CHAR(13)
                                           INIT('  Message :  '),
               05 SM_MSG_DATA             CHAR(66);

        DCL
            01 DISP_SERVER_MSG_5X,
               05 FILLER                  CHAR(13) INIT(' '),
               05 SM_MSG_DATA_X           CHAR(66);

/*-------------------------------------------------------------------*/
/* CICS Condition Handler                                            */
/*-------------------------------------------------------------------*/

        EXEC CICS HANDLE CONDITION MAPFAIL(NO_INPUT)
                                   ERROR(ERRORS) ;

/*-------------------------------------------------------------------*/
/* CICS Aid Handler                                                  */
/*-------------------------------------------------------------------*/

        EXEC CICS HANDLE AID ANYKEY(NO_INPUT)
                             CLEAR(GETOUT) ;

/*-------------------------------------------------------------------*/
/* program initialization                                            */
/*-------------------------------------------------------------------*/

        DIAG_MSGS_INITIALIZED = TRUE ;
        MSG_TEXT_2            = 'Press Clear To Exit';
        NO_ERRORS_SW          = TRUE ;
        PAGE_CNT              = PAGE_CNT + 1;
        SERVERL               = -1 ;

        DO I1 = 1 TO 13 ;
          RSLTNO( I1 ) = BLANK ;
        END ;

        CALL GET_SYSTEM_TIME ;

GET_INPUT_AGAIN:

          CALL DISPLAY_INITIAL_SCREEN ;
          CALL GET_INPUT_DATA ;
```

```
/*-------------------------------------------------------------------*/
/* allocate a context structure                                      */
/*-------------------------------------------------------------------*/

        CALL CSBCTXAL( CS_VERSION_50,
                       CSL_RC,
                       CSL_CTX_HANDLE );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CSCTXALLOC failed';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;

/*-------------------------------------------------------------------*/
/* initialize the Client-Library                                     */
/*-------------------------------------------------------------------*/

        CALL CTBINIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_VERSION_50 );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR       = 'CTBINIT failed';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;

        CALL PROCESS_INPUT ;

        CALL QUIT_CLIENT_LIBRARY ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to get system time                                     */
/*                                                                   */
/*-------------------------------------------------------------------*/
GET_SYSTEM_TIME: PROC ;

        EXEC CICS ASKTIME ABSTIME(UTIME);

        EXEC CICS FORMATTIME
```

```
                    ABSTIME(UTIME)
                    DATESEP('/')
                    MMDDYY(TMP_DATE)
                    TIME(TMP_TIME)
                    TIMESEP ;

END GET_SYSTEM_TIME ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to get system time                                     */
/*                                                                   */
/*-------------------------------------------------------------------*/
DISPLAY_INITIAL_SCREEN: PROC ;

        SDATEO   = TMP_DATE ;
        STIMEO   = TMP_TIME ;
        MSG1O    = MSG_TEXT_1 ;
        PROGNMO  = 'SYCTSAA4' ;
        MSG1O    = MSG_TEXT_1 ;
        MSG2O    = MSG_TEXT_2 ;
        SPAGEO   = '0001' ;

        EXEC CICS SEND MAP('A4PANEL')
                       MAPSET('SYCTBA4')
                       CURSOR
                       FRSET
                       ERASE
                       FREEKB ;

END DISPLAY_INITIAL_SCREEN ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to get input data                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/
GET_INPUT_DATA: PROC ;

        EXEC CICS RECEIVE MAP('A4PANEL')
                          MAPSET('SYCTBA4')
                          ASIS ;

        IF SERVERL = 0 THEN
        DO ;
          IF PF_SERVER = BLANK THEN
```

```
  DO ;
    SERVERL        = -1 ;        /* set the cursor position */
    MSG_TEXT_1     = 'Please Enter Server Name' ;
    ENTER_DATA_SW  = TRUE ;
  END ;
END ;
ELSE DO ;
  PF_SERVER      = SERVERI ;
  PF_SERVER_SIZE = SERVERL ;
END ;

IF USERL = 0 THEN
DO ;
  IF PF_USER = BLANK THEN
  DO ;
    USERL          = -1 ;        /* set the cursor position */
    MSG_TEXT_1     = 'Please Enter User-ID' ;
    ENTER_DATA_SW  = TRUE ;
  END ;
END ;
ELSE DO ;
  PF_USER      = USERI ;
  PF_USER_SIZE = USERL ;
END ;

IF PSWDL ^= 0 THEN
DO ;
  PF_PWD      = PSWDI;
  PF_PWD_SIZE = PSWDL ;
END ;

IF TRANL ^= 0 THEN
DO ;
  PF_TRAN     = TRANI;
  PF_TRANL = TRANL ;
END ;

IF NETDRVL ^= 0 THEN
DO ;
  PF_NETDRV   = NETDRVI ;
  PF_DRV_SIZE = NETDRVL ;
END ;

IF ENTER_DATA_SW = TRUE THEN
  DO ;
    ENTER_DATA_SW = FALSE ;
```

```
            CALL DISPLAY_INITIAL_SCREEN ;
            MSG_TEXT_1     = BLANK ;
            CALL GET_INPUT_DATA ;
          END ;

 END GET_INPUT_DATA ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process input data                                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
 PROCESS_INPUT: PROC ;

 /*---------------------------------------------------------------*/
 /* allocate a connection to the server                           */
 /*---------------------------------------------------------------*/

         CSL_CON_HANDLE = 0 ;

         CALL CTBCONAL( CSL_CTX_HANDLE,
                        CSL_RC,
                        CSL_CON_HANDLE ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBCONALLOC failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

 /*---------------------------------------------------------------*/
 /* alter properties of the connection for user-id                */
 /*---------------------------------------------------------------*/

         CALL CTBCONPR( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_SET,
                        CS_USERNAME,
                        PF_USER,
                        PF_USER_SIZE,
                        CS_FALSE,
                        OUTLEN ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
```

```
        DO ;
          MSGSTR = 'CTBCONPROPS for user-id failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for password             */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_PASSWORD,
                       PF_PWD,
                       PF_PWD_SIZE,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for password failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for transaction          */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_TRANSACTION_NAME,
                       PF_TRAN,
                       PF_TRANL,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for transaction failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
```

```
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for Network driver       */
/*-------------------------------------------------------------*/

        SELECT;
          WHEN (PF_NETDRV = '        ')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'LU62' | PF_NETDRV = 'lu62')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'IBMTCPIP' | PF_NETDRV = 'ibmtcpip')
                NETDRIVER = CS_TCPIP ;
          WHEN (PF_NETDRV = 'INTERLIN' | PF_NETDRV = 'interlin')
                NETDRIVER = CS_INTERLINK ;
          WHEN (PF_NETDRV = 'CPIC' | PF_NETDRV = 'cpic')
                NETDRIVER = CS_NCPIC ;
          OTHERWISE
            DO;
                MSGSTR = 'Invalid Network driver entered';
                NO_ERRORS_SW = FALSE ;
                CALL ERROR_OUT;
                CALL ALL_DONE ;
            END;
        END;

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_NET_DRIVER,
                       NETDRIVER,
                       CS_UNUSED,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for Network driver failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* setup retrieval of All Messages                             */
```

```
/*-------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_INIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      CS_UNUSED ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_INIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* set the upper limit of number of messages                   */
/*-------------------------------------------------------------*/

        PF_MSGLIMIT = 5 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_MSGLIMIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      PF_MSGLIMIT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_MSGLIMIT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* open connection to the server or CICS region                */
/*-------------------------------------------------------------*/

        CALL CTBCONNE( CSL_CON_HANDLE,
                       CSL_RC,
```

```
                        PF_SERVER,
                        PF_SERVER_SIZE,
                        CS_FALSE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONNECT failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* invokes SEND_COMMAND routine                                */
/*-------------------------------------------------------------*/
        IF NO_ERRORS_SW
          THEN
            CALL SEND_COMMAND ;

/*-------------------------------------------------------------*/
/* process the results of the command                          */
/*-------------------------------------------------------------*/

        IF NO_ERRORS_SW THEN
        DO ;
          DO WHILE( ^NO_MORE_RESULTS ) ;
            CALL PROCESS_RESULTS ;
          END ;
          CALL CLOSE_CONNECTION ;
        END ;

END PROCESS_INPUT ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to allocate, send, and process commands                */
/*                                                                   */
/*-------------------------------------------------------------------*/
SEND_COMMAND: PROC ;

/*-------------------------------------------------------------*/
/* find out what the maximum number of connections is          */
/*-------------------------------------------------------------*/

        CALL CTBCONFI( CSL_CTX_HANDLE,
                       CSL_RC,
```

```
                        CS_GET,
                        CS_MAX_CONNECT,
                        CF_MAXCONNECT,
                        STG(CF_MAXCONNECT),
                        CS_FALSE,
                        CF_OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCONFIG failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* display number of connections                               */
/*-------------------------------------------------------------*/

        OR2_MAXCONNECT = CF_MAXCONNECT;

/*-------------------------------------------------------------*/
/* allocate a command handle                                   */
/*-------------------------------------------------------------*/

        CALL CTBCMDAL( CSL_CON_HANDLE,
                       CSL_RC,
                       CSL_CMD_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCMDALLOC failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* prepare the language request                                */
/*-------------------------------------------------------------*/

        PF_STRLEN = STG(CF_LANG2 ) ;

        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_LANG_CMD,
```

```
                        CF_LANG2,
                        PF_STRLEN,
                        CS_UNUSED ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBCOMMAND failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-----------------------------------------------------------------*/
/* send the language request                                       */
/*-----------------------------------------------------------------*/

        CALL CTBSEND( CSL_CMD_HANDLE,
                      CSL_RC ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBSEND failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

END SEND_COMMAND ;

/*-----------------------------------------------------------------------*/
/*                                                                       */
/* Subroutine to process the result                                      */
/*                                                                       */
/*-----------------------------------------------------------------------*/
PROCESS_RESULTS: PROC ;

/*-----------------------------------------------------------------*/
/* set up the results data                                         */
/*-----------------------------------------------------------------*/

        CALL CTBRESUL( CSL_CMD_HANDLE,
                       CSL_RC,
                       RF_TYPE ) ;

/*-----------------------------------------------------------------*/
/* determine the outcome of the comand execution                   */
```

```
/*-------------------------------------------------------------*/

        SELECT( CSL_RC ) ;

          WHEN( CS_SUCCEED )
          DO ;

/*-------------------------------------------------------------*/
/* determine the type of result returned by the current request */
/*-------------------------------------------------------------*/

            SELECT( RF_TYPE ) ;

/*-------------------------------------------------------------*/
/* process row results                                         */
/*-------------------------------------------------------------*/

              WHEN( CS_ROW_RESULT )
              DO ;
                CALL RESULT_ROW_PROCESSING ;
                DO WHILE( ^NO_MORE_ROWS ) ;
                  CALL FETCH_ROW_PROCESSING ;
                END ;
              END ;

/*-------------------------------------------------------------*/
/* process parameter results --- there should be no parameter */
/* to process                                                  */
/*-------------------------------------------------------------*/

              WHEN( CS_PARAM_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;
              END ;

/*-------------------------------------------------------------*/
/* process status results --- the stored procedure status     */
/* result will not be processed in this example               */
/*-------------------------------------------------------------*/

              WHEN( CS_STATUS_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;
              END ;

/*-------------------------------------------------------------*/
```

```
/* print an error message if the server encountered an error  */
/* while executing the request                                */
/*------------------------------------------------------------*/

              WHEN( CS_CMD_FAIL )
              DO ;
                NO_ERRORS_SW = FALSE ;
                MSGSTR       =
                  'CTBRESUL returned CS_CMD-FAIL restype' ;
                CALL ERROR_OUT ;
              END ;

/*------------------------------------------------------------*/
/* print a message for successful commands that returned no   */
/* data( optional )                                           */
/*------------------------------------------------------------*/

              WHEN( CS_CMD_SUCCEED )
              DO ;
                MSGSTR = 'CTBRESUL returned CS_CMD_SUCCEED restype' ;
              END ;

/*------------------------------------------------------------*/
/* print a message for requests that have been processed      */
/* successfully( optional )                                   */
/*------------------------------------------------------------*/

              WHEN( CS_CMD_DONE )
              DO ;
                MSGSTR = 'CTBRESUL returned CS_CMD_DONE restype' ;
              END ;

              OTHERWISE
              DO ;
                NO_MORE_RESULTS = TRUE ;
                NO_ERRORS_SW    = FALSE ;
                MSGSTR          = 'CTBRESUL returned UNKNOWN restype' ;
                CALL ERROR_OUT ;

              END ;
            END ; /* end of SELECT( RF_TYPE ) */
          END ;

/*------------------------------------------------------------*/
/* print an error message if the CTBRESULTS call failed       */
/*------------------------------------------------------------*/
```

```
          WHEN( CS_FAIL )
          DO ;
            NO_MORE_RESULTS = TRUE ;
            NO_ERRORS_SW    = FALSE ;
            MSGSTR          = 'CTBRESUL returned CS_FAIL ret_code' ;
            CALL ERROR_OUT ;
          END ;

/*-------------------------------------------------------------*/
/* drop out of the results loop if no more result sets are     */
/* available for processing or if the results were cancelled   */
/*-------------------------------------------------------------*/

          WHEN( CS_END_RESULTS )
          DO ;
            NO_MORE_RESULTS = TRUE ;
          END ;

          WHEN( CS_CANCELLED )
          DO ;
            NO_MORE_RESULTS = TRUE ;
          END ;

          OTHERWISE
          DO ;
            NO_MORE_RESULTS = TRUE ;
            NO_ERRORS_SW    = FALSE ;
            MSGSTR          =
               'CTBRESUL returned unknown ret_code' ;
            CALL ERROR_OUT ;
          END ;
        END ; /* end of SELECT( CSL_RC ) */

        RF_TYPE = 0 ;

END PROCESS_RESULTS ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process result rows                                 */
/*                                                                   */
/*-------------------------------------------------------------------*/
RESULT_ROW_PROCESSING: PROC ;

/*-------------------------------------------------------------*/
```

```
/* We need to bind the data to program variables. We don't    */
/* care about the indicator variable so we'll pass NULL for   */
/* that parameter in OC_BIND().                               */
/*------------------------------------------------------------*/

        CALL CTBRESIN( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_NUMDATA,
                       RF_NUMDATA,
                       STG(RF_NUMDATA),
                       CF_COL_LEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBRESINFO failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

        FF_ROW_NUM = FF_ROW_NUM + 1;

/*------------------------------------------------------------*/
/* display the number of connections                          */
/*------------------------------------------------------------*/

        OR2_MAXCONNECT     = CF_MAXCONNECT ;
        RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR2 ;
        FF_ROW_NUM         = FF_ROW_NUM + 2;

/*------------------------------------------------------------*/
/* display the number of columns                              */
/*------------------------------------------------------------*/

        OR4_NUMDATA        = RF_NUMDATA ;
        RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR4 ;

        IF RF_NUMDATA ^= 2 THEN
        DO ;
          MSGSTR       = 'CTBRESINFO returned wrong # of parms' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

        FF_ROW_NUM = FF_ROW_NUM + 2;
```

```
/*-------------------------------------------------------------*/
/* Setup column headings                                       */
/*-------------------------------------------------------------*/

        RSLTNO(FF_ROW_NUM) = 'FirstName    EducLvl' ;
        FF_ROW_NUM         = FF_ROW_NUM + 1;
        RSLTNO(FF_ROW_NUM) = '===========  =======' ;

        DO PARM_CNT = 1 TO RF_NUMDATA  ;
          CALL BIND_COLUMNS ;
        END ;

END RESULT_ROW_PROCESSING ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to bind each data                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/
BIND_COLUMNS: PROC ;

        CALL CTBDESCR( CSL_CMD_HANDLE,
                       CSL_RC,
                       PARM_CNT,
                       DATAFMT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBDESCRIBE failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

/*-------------------------------------------------------------*/
/* We need TO bind the data TO program variables. We don't     */
/* care about the indicator variable so we'll pass NULL for    */
/* that PARAMeter in OC_BIND().                                */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/* rows per fetch                                              */
/*-------------------------------------------------------------*/

        DF_COUNT = 1 ;
```

```
        SELECT( DF_DATATYPE ) ;

/*-------------------------------------------------------------*/
/* bind the first column, FIRSTNME defined as VARCHAR(12)      */
/*-------------------------------------------------------------*/
          WHEN( CS_VARCHAR_TYPE )
          DO ;
            DF_DATATYPE   = CS_VARCHAR_TYPE;
            DF_FORMAT     = CS_FMT_UNUSED;
            DF_MAXLENGTH  = STG(CF_COL_FIRSTNME) - 2;
            DF_COUNT      = 1;
            CF_COL_NUMBER = 1;

            CALL CTBBIND( CSL_CMD_HANDLE,
                          CSL_RC,
                          CF_COL_NUMBER,
                          DATAFMT,
                          CF_COL_FIRSTNME,
                          CF_COL_LEN,
                          CS_PARAM_NOTNULL,
                          CF_COL_INDICATOR,
                          CS_PARAM_NULL);

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBBIND CS_VARCHAR_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
            END ;
          END ;

/*-------------------------------------------------------------*/
/* bind the second column, EDLEVEL defined as SMALLINT         */
/*-------------------------------------------------------------*/
          WHEN( CS_SMALLINT_TYPE )
          DO ;
            DF_DATATYPE   = CS_SMALLINT_TYPE;
            DF_FORMAT     = CS_FMT_UNUSED;
            DF_MAXLENGTH  = STG(CF_COL_EDLEVEL);
            DF_COUNT      = 1;
            CF_COL_NUMBER = 2;

            CALL CTBBIND( CSL_CMD_HANDLE,
                          CSL_RC,
```

```
                        CF_COL_NUMBER,
                        DATAFMT,
                        CF_COL_EDLEVEL,
                        CF_COL_LEN,
                        CS_PARAM_NOTNULL,
                        CF_COL_INDICATOR,
                        CS_PARAM_NULL ) ;

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CTBBIND CS_SMALLINT_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
            END ;
          END ;

          OTHERWISE ;

        END ; /* end of SELECT( DF_DATATYPE ) */

END BIND_COLUMNS ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to fetch row processing                                 */
/*                                                                    */
/*--------------------------------------------------------------------*/
FETCH_ROW_PROCESSING: PROC ;

        CALL CTBFETCH( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_UNUSED,       /* type   */
                       CS_UNUSED,       /* offset */
                       CS_UNUSED,       /* option */
                       FF_ROWS_READ ) ;

        SELECT( CSL_RC ) ;

          WHEN( CS_SUCCEED )
          DO ;
            NO_MORE_ROWS         = FALSE ;
            CF_COL_FIRSTNME_CHAR = BLANK ;
            DF_DATATYPE          = CS_VARCHAR_TYPE;
            DF_MAXLENGTH         = LENGTH( CF_COL_FIRSTNME ) ;
            DF2_DATATYPE         = CS_CHAR_TYPE;
```

```
            DF2_MAXLENGTH        = STG(CF_COL_FIRSTNME_CHAR);

            CALL CSBCONVE( CSL_CTX_HANDLE,
                           CSL_RC,
                           DATAFMT,
                           CF_COL_FIRSTNME,
                           DATAFMT2,
                           CF_COL_FIRSTNME_CHAR,
                           CF_COL_LEN);

            IF CSL_RC ^= CS_SUCCEED THEN
            DO ;
              MSGSTR       = 'CSCONVERT CS_CHAR_TYPE failed' ;
              NO_ERRORS_SW = FALSE ;
              CALL ERROR_OUT;
              CALL ALL_DONE;
            END ;

            FF_ROW_NUM = FF_ROW_NUM + 1 ;

/*-------------------------------------------------------------*/
/* save ROW RESULTS for later display                          */
/*-------------------------------------------------------------*/

            OR_COL_FIRSTNME_CHAR = CF_COL_FIRSTNME_CHAR;
            OR_COL_EDLEVEL       = CF_COL_EDLEVEL;

            IF FF_ROW_NUM > 10 THEN
            DO;
              MSG_TEXT_1 = 'Please press return to continue!' ;
              MSG_TEXT_2 = BLANK ;
              CALL DISP_DATA ;
              FF_ROW_NUM = 1;
              PAGE_CNT = PAGE_CNT + 1 ;

/*-------------------------------------------------------------*/
/* Setup column headings                                       */
/*-------------------------------------------------------------*/

              RSLTNO(FF_ROW_NUM) = 'FirstName    EducLvl' ;
              FF_ROW_NUM         = FF_ROW_NUM + 1 ;
              RSLTNO(FF_ROW_NUM) = '==========  =======' ;
              FF_ROW_NUM         = FF_ROW_NUM + 1 ;
            END ;

            RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR;
```

```
END ; /* end of WHEN( CS_SUCCEED ) */

WHEN( CS_END_DATA )
DO ;
  NO_MORE_ROWS = TRUE ;
  MSG_TEXT_1   = 'All rows processing completed!' ;
  MSG_TEXT_2   = 'Press Clear To Exit';
  CALL DISP_DATA ;
END ; /* end of WHEN( CS_END_DATA ) */

WHEN( CS_FAIL )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
       'CTBFETCH returned CS_FAIL ret_code' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_FAIL ) */

WHEN( CS_ROW_FAIL )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
       'CTBFETCH returned CS_ROW_FAIL ret_code' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_ROW_FAIL ) */

WHEN( CS_CANCELLED )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSG1O     = 'CTBFETCH returned CS_CANCELLED ret_code' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_CANCELLED ) */

OTHERWISE
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
       'CTBFETCH returned Unknown ret_code' ;
  CALL ERROR_OUT;
END ; /* end of OTHERWISE */
```

```
        END ; /* end of SELECT( CSL_RC ) */

END FETCH_ROW_PROCESSING ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to print output messages.                               */
/*                                                                    */
/*--------------------------------------------------------------------*/

ERROR_OUT: PROC;

        SAMP_RC   = CSL_RC;
        REST_TYPE = RF_TYPE ;

        IF DIAG_MSGS_INITIALIZED
          THEN
            CALL GET_DIAG_MESSAGES ;

/*--------------------------------------------------------------------*/
/* display error messages                                             */
/*--------------------------------------------------------------------*/

        MSG_TEXT_1 = TEST_CASE || SAMP_LIT  || SAMP_RC ||
                     REST_LIT  || REST_TYPE || '   '   ||
                     MSGSTR ;

        IF PRINT_ONCE THEN
          DO ;
            CALL DISP_DATA ;
            PRINT_ONCE = FALSE ;
          END ;

        NO_ERRORS_SW = FALSE ;
        MSGSTR       = BLANK ;
        SAMP_RC      = 0;
        REST_TYPE    = 0 ;

END ERROR_OUT;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve any diagnostic messages                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
GET_DIAG_MESSAGES: PROC ;
```

```
 DCL CNT        FIXED BIN(15) ;
/*-------------------------------------------------------------------*/
/* Disable calls to this subroutine                                  */
/*-------------------------------------------------------------------*/

         DIAG_MSGS_INITIALIZED = FALSE ;

 /*------------------------------------------------------------------*/
 /* First, get client messages                                       */
 /*------------------------------------------------------------------*/

         CALL CTBDIAG( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_UNUSED,
                       CS_STATUS,
                       CS_CLIENTMSG_TYPE,
                       CS_UNUSED,
                       DF_NUM_OF_MSGS ) ;

         IF CSL_RC ^= CS_SUCCEED THEN
           DO ;
             MSGSTR = 'CTBDIAG CS_STATUS CLIENTMSG_TYPE failed';
             CALL ERROR_OUT ;
             CALL ALL_DONE ;
           END ;
           ELSE DO ;
             IF DF_NUM_OF_MSGS > 0 THEN
               DO ;
                 DO CNT = 1 TO DF_NUM_OF_MSGS ;
                   CALL RETRIEVE_CLIENT_MSGS ;
                 END ;
               END ;
           END ;

 /*------------------------------------------------------------------*/
 /* Then, get server messages                                        */
 /*------------------------------------------------------------------*/

         CALL CTBDIAG( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_UNUSED,
                       CS_STATUS,
                       CS_SERVERMSG_TYPE,
                       CS_UNUSED,
                       DF_NUM_OF_MSGS ) ;
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_STATUS SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;
          ELSE DO ;
            IF DF_NUM_OF_MSGS > 0 THEN
              DO ;
                DO CNT = 1 TO DF_NUM_OF_MSGS ;
                  CALL RETRIEVE_SERVER_MSGS ;
                END ;
              END ;
          END ;

END GET_DIAG_MESSAGES ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve diagnostic messages from client             */
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_CLIENT_MSGS: PROC ;

        I1 = 1 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
                      CS_CLIENTMSG_TYPE,
                      DF_MSGNO,
                      CLIENT_MSG ) ;

        IF CSL_RC ^=  CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_GET CS_CLIENTMSG_TYPE FAILED' ;
          CALL ERROR_OUT ;
          CALL ALL_DONE ;
        END ;

/*--------------------------------------------------------------------*/
/* display message text                                               */
/*--------------------------------------------------------------------*/
```

```
RSLTNO( I1 )      = 'Client Message:' ;
I1                = 3 ;

CM_SEVERITY_DATA = CM_SEVERITY ;
CM_STATUS_DATA   = CM_STATUS ;
RSLTNO( I1 )      = CM_SEVERITY_HDR || CM_SEVERITY_DATA ||
                    CM_STATUS_HDR   || CM_STATUS_DATA ;
I1                = I1 + 1 ;

CM_OC_MSGNO_DATA = CM_MSGNO ;
RSLTNO( I1 )      = CM_OC_MSGNO_HDR || CM_OC_MSGNO_DATA ;
I1                = I1 + 1 ;

IF CM_MSGNO ^= 0 THEN
DO ;
    CM_OC_MSG_DATA     = SUBSTR( CM_TEXT, 1, 66 ) ;
    RSLTNO( I1 )       = '  OC MsgTx:  ' || CM_OC_MSG_DATA ;
    I1                 = I1 + 1 ;
    IF CM_TEXT_LEN > 66 THEN
    DO ;
        CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 67, 66 ) ;
        RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
        I1               = I1 + 1 ;
        IF CM_TEXT_LEN > 132 THEN
        DO ;
            CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 133, 66 ) ;
            RSLTNO( I1 )     = BLANK_13 ||
                               CM_OC_MSG_DATA_X ;
            I1               = I1 + 1 ;
            IF CM_TEXT_LEN > 198 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 199 ) ;
                RSLTNO( I1 )     = BLANK_13 ||
                                   CM_OC_MSG_DATA_X ;
                I1               = I1 + 1 ;
            END ;
        END ;
    END ;
END ;
ELSE DO ;
  RSLTNO( I1 ) = '  OC MsgTx:  No Message!' ;
  I1           = I1 + 1 ;
END ;

CM_OS_MSGNO_DATA = CM_OS_MSGNO ;
RSLTNO( I1 )      = '  OS MsgNo:  ' || CM_OS_MSGNO_DATA ;
```

```
        I1                = I1 + 1 ;

        IF CM_OS_MSGNO ^= 0 THEN
        DO ;
            CM_OS_MSG_DATA      = SUBSTR( CM_OS_MSGTXT, 1, 66 ) ;
            RSLTNO( I1 )        = '  OS MsgTx:  ' ||
                                  CM_OS_MSG_DATA ;
            I1                  = I1 + 1 ;
            IF CM_OS_MSGTEXT_LEN > 66 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT, 67, 66 ) ;
                RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
                I1               = I1 + 1 ;
                IF CM_OS_MSGTEXT_LEN > 132 THEN
                DO ;
                    CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,133,66 );
                    RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
                    I1               = I1 + 1 ;
                    IF CM_OS_MSGTEXT_LEN > 198 THEN
                    DO ;
                        CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,199 );
                        RSLTNO( I1 )     = BLANK_13 ||
                                           CM_OC_MSG_DATA_X ;
                        I1               = I1 + 1 ;
                    END ;
                END ;
            END ;
        END ;
        ELSE DO ;
          RSLTNO( I1 ) = '  OS MsgTx:  No Message!' ;
          I1           = I1 + 1 ;
        END ;

END RETRIEVE_CLIENT_MSGS ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve diagnostic messages from server             */
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_SERVER_MSGS: PROC ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
```

```
                    CS_SERVERMSG_TYPE,
                    DF_MSGNO,
                    SERVER_MSG ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_GET CS_SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;

/*-------------------------------------------------------------------*/
/* display message text                                              */
/*-------------------------------------------------------------------*/

        SM_MSG_NO_DATA     = SM_MSGNO ;
        SM_SEVERITY_DATA   = SM_SEV ;
        SM_STATE_DATA      = SM_STATE ;
        SM_LINE_NO_DATA    = SM_LINE ;
        SM_STATUS_DATA     = SM_STATUS ;

        IF SM_SVRNAME_LEN > 66
          THEN
            SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 63 ) || '...' ;
          ELSE
            SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 66 ) ;

        IF SM_PROC_LEN > 66
          THEN
            SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 63 ) || '...' ;
          ELSE
            SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 66 ) ;

        SM_MSG_DATA        = SUBSTR( SM_TEXT, 1, 66 ) ;
        RSLTNO (1)         = 'Server Message:' ;
        RSLTNO (3)         = SM_MSG_NO_HDR   || SM_MSG_NO_DATA ||
                             SM_SEVERITY_HDR || SM_SEVERITY_DATA ||
                             SM_STATE_HDR    || SM_STATE_DATA ;
        RSLTNO (4)         = SM_LINE_NO_HDR  || SM_LINE_NO_DATA ||
                             SM_STATUS_HDR   || SM_STATUS_DATA ;
        RSLTNO (5)         = SM_SVRNAME_HDR  || SM_SVRNAME_DATA ;
        RSLTNO (6)         = SM_PROC_ID_HDR  || SM_PROC_ID_DATA ;
        RSLTNO (7)         = SM_MSG_HDR      || SM_MSG_DATA ;

        IF SM_TEXT_LEN > 66 THEN
          DO ;
```

```
            SM_MSG_DATA_X = SUBSTR( SM_TEXT, 67, 66 ) ;
            RSLTNO(8)     = BLANK_13 || SM_MSG_DATA_X ;
            IF SM_TEXT_LEN > 132 THEN
              DO ;
                SM_MSG_DATA_X = SUBSTR( SM_TEXT, 133, 66 ) ;
                RSLTNO(9)     = BLANK_13 || SM_MSG_DATA_X ;
                IF SM_TEXT_LEN > 198 THEN
                  DO ;
                    SM_MSG_DATA_X = SUBSTR( SM_TEXT, 198 ) ;
                    RSLTNO(10)    = BLANK_13 || SM_MSG_DATA_X ;
                  END ;
              END ;
          END ;

 END RETRIEVE_SERVER_MSGS ;

 /*--------------------------------------------------------------------*/
 /*                                                                    */
 /* Subroutine to drop and to deallocate all handlers, to close        */
 /* server connection and exit client library                          */
 /*                                                                    */
 /*--------------------------------------------------------------------*/
 ALL_DONE: PROC ;

         CALL CLOSE_CONNECTION;
         CALL QUIT_CLIENT_LIBRARY;
         STOP ;

 END ALL_DONE ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to perform drop command handler, close server           */
/* connection, and deallocate Connection Handler.                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
 CLOSE_CONNECTION: PROC ;

 /*--------------------------------------------------------------------*/
 /* drop the command handle                                            */
 /*--------------------------------------------------------------------*/

         CALL CTBCMDDR( CSL_CMD_HANDLE,
                        CSL_RC ) ;

         IF CSL_RC = CS_FAIL THEN
```

```
        DO ;
          MSGSTR = 'CTBCMDDROP failed' ;
          CALL ERROR_OUT ;
        END ;

 /*-------------------------------------------------------------------*/
 /* close the server connection                                       */
 /*-------------------------------------------------------------------*/

        CALL CTBCLOSE( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCLOSE failed' ;
          CALL ERROR_OUT ;
        END ;

 /*-------------------------------------------------------------------*/
 /* DE_ALLOCATE THE CONNECTION HANDLE                                 */
 /*-------------------------------------------------------------------*/

        CALL CTBCONDR( CSL_CON_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCONDROP failed' ;
          CALL ERROR_OUT ;
        END ;

 END CLOSE_CONNECTION ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to perform exit client library and deallocate context */
/* structure.                                                        */
/*                                                                   */
/*-------------------------------------------------------------------*/
 QUIT_CLIENT_LIBRARY: PROC ;

 /*-------------------------------------------------------------------*/
 /* exit the Client Library                                           */
 /*-------------------------------------------------------------------*/
```

```
        CALL CTBEXIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_UNUSED ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBEXIT failed' ;
          CALL ERROR_OUT ;
        END ;

/*-------------------------------------------------------------------*/
/* de-allocate the context structure                                 */
/*-------------------------------------------------------------------*/

        CALL CSBCTXDR( CSL_CTX_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CSBCTXDROP failed' ;
          CALL ERROR_OUT ;
        END ;

        EXEC CICS RETURN ;

END QUIT_CLIENT_LIBRARY ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to display output                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/
DISP_DATA: PROC ;

        SDATEO  = TMP_DATE ;
        STIMEO  = TMP_TIME;
        PROGNMO = 'SYCTSAA4' ;

        SELECT( PAGE_CNT ) ;
          WHEN( 1 ) SPAGEO = '0001' ;
          WHEN( 2 ) SPAGEO = '0002' ;
          WHEN( 3 ) SPAGEO = '0003' ;
          WHEN( 4 ) SPAGEO = '0004' ;
          WHEN( 5 ) SPAGEO = '0005' ;
          WHEN( 6 ) SPAGEO = '0006' ;
          WHEN( 7 ) SPAGEO = '0007' ;
```

```
          WHEN( 8 ) SPAGEO = '0008' ;
          WHEN( 9 ) SPAGEO = '0009' ;
          OTHERWISE SPAGEO = '9999' ;
        END ;

        SERVERA = DFHBMPRO;
        SERVERO = PF_SERVER;

        USERA   = DFHBMPRO;
        USERO   = PF_USER;

        NETDRVA   = DFHBMPRO;
        NETDRVO   = PF_NETDRV;

        PSWDA   = DFHBMDAR;
        PSWDO   = PF_PWD;
        MSG1O   = MSG_TEXT_1;
        MSG2O   = MSG_TEXT_2;

/*-------------------------------------------------------------------*/
/* DISPLAY THE DATA                                                  */
/*-------------------------------------------------------------------*/

        EXEC CICS SEND MAP('A4PANEL')
                       MAPSET('SYCTBA4')
                       CURSOR
                       FRSET
                       ERASE
                       FREEKB ;

        EXEC CICS RECEIVE INTO(QF_ANSWER)
                          LENGTH(QF_LEN)
                          MAXLENGTH(QF_MAXLEN)
                          RESP(CICS_RESPONSE) ;

END DISP_DATA ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: NO_INPUT --- to handle MAPFAIL/ANYKEY condition            */
/*                                                                   */
/*-------------------------------------------------------------------*/
NO_INPUT:

        MSG_TEXT_1 = 'Please Enter Input Fields' ;
        GO TO GET_INPUT_AGAIN ;
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: GETOUT --- to handle CLEAR condition                       */
/*                                                                   */
/*-------------------------------------------------------------------*/
GETOUT:

        EXEC CICS RETURN ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: ERRORS --- to handle ERROR condition                       */
/*                                                                   */
/*-------------------------------------------------------------------*/
ERRORS:

        EXEC CICS DUMP DUMPCODE('ERRS') ;

END SYCTSAA4;
```

APPENDIX B

Sample RPC Application

This appendix contains a sample Open ClientConnect application program, SYCTSAR4, that sends an RPC to either Open ServerConnect or Adaptive Server Enterprise.

The purpose of this sample program is to demonstrate the use of Client-Library functions, particularly those designed to send RPC requests. In some cases, one Client-Library function is used for demonstration purposes when another function would be more efficient. In order to best illustrate the flow of processing, the program does not do extensive error checking.

The remote procedure or transaction initiated by this RPC is called SYR2. SYR2 uses data from the sample table SYBASE.SAMPLETB.

- If your server is an Adaptive Server Enterprise, the remote procedure and table must be created by Transaction Router Service (TRS). A script is provided with TRS that it can use to create SYR2 and the sample table on Adaptive Server Enterprise.
- If your remote server is Open ServerConnect, SYR2 and SYBASE.SAMPLETB are provided on the product API.

SYCTSAR4 is provided as part of the Open ClientConnect package.

Sample program – SYCTSAR4

```
SYCTSAR4: PROC OPTIONS(MAIN REENTRANT)              /*       @(#) syctsar4.pli
1.1 5/8/96     */

 /****** SYCTSAR4 - Client RPC Request APPL - PL/I - CICS ************/
 /*                                                                  */
 /*  CICS TRANID:    SYR4                                            */
 /*                                                                  */
 /*  PROGRAM:       SYCTSAR4                                         */
 /*                                                                  */
 /*  PURPOSE:  Demonstrates Open Client for CICS CALLs.              */
```

```
/*                                                                  */
/*  FUNCTION: Illustrates how to send an RPC request with           */
/*            parameters to:                                        */
/*                                                                  */
/*              - A SQL Server                                      */
/*              - An Open Server running in a CICS region.          */
/*                                                                  */
/*            SQL Server:                                           */
/*                                                                  */
/*              If the request is sent to a SQL Server it           */
/*              intitiates the stored procedure "SYR2".             */
/*                                                                  */
/*               Note: The Net-Gateway/MCG product includes a script*/
/*                     that creates this procedure in a target SQL  */
/*                     server.                                      */
/*                                                                  */
/*            Open Server/CICS:                                     */
/*                                                                  */
/*              If the request is sent to an Open Server/CICS       */
/*              region, invoke the CICS transaction SYR1.           */
/*                                                                  */
/*              Note: The Open Server/CICS product includes the     */
/*                    sample transaction SYR1. This is the server   */
/*                    side transaction invoked by this program.     */
/*                                                                  */
/*            Open Server/IMS:                                      */
/*                                                                  */
/*              If the request is sent to an Open Server/IMS        */
/*              region, invoke the IMS transaction SYR1.            */
/*                                                                  */
/*              Note: The Open Server/IMS product includes the      */
/*                    sample transaction SYR1. This is the server   */
/*                    side transaction invoked by this program.     */
/*                                                                  */
/*  PREREQS:  Before running SYCTSAR4, make sure that the server    */
/*            you wish to access has an entry in the Connection     */
/*            Router Table for that Server and the MCG(s) that      */
/*            you wish to use.                                      */
/*                                                                  */
/*  INPUT:    On the input screen, make sure to enter the Server    */
/*            name, user id, and password for the target server.    */
/*            TRAN NAME is not used for LAN servers.                */
/*                                                                  */
/*            If the target server is in a CICS or IMS region,      */
/*            enter SYR1 in the TRAN NAME field if the server is    */
/*                                                                  */
```

```
/*  Open Client CALLs used in this sample:                          */
/*                                                                  */
/*    CSBCTXALLOC   allocate a context                              */
/*    CSBCTXDROP    drop a context                                  */
/*    CTBBIND       bind a column variable                          */
/*    CTBCLOSE      close a server connection                       */
/*    CTBCMDALLOC   allocate a command                              */
/*    CTBCMDDROP    drop a command                                  */
/*    CTBCOMMAND    initiate remote procedure call                  */
/*    CTBCONALLOC   allocate a connection                           */
/*    CTBCONDROP    drop a connection                               */
/*    CTBCONPROPS   alter properties of a connection                */
/*    CTBCONNECT    open a server connection                        */
/*    CTBDESCRIBE   return a description of result data             */
/*    CTBDIAG       retrieve SQLCODE messages                       */
/*    CTBEXIT       exit client library                             */
/*    CTBFETCH      fetch result data                               */
/*    CTBINIT       init client library                             */
/*    CTBPARAM      define a command parameter                      */
/*    CTBRESULTS    sets up result data                             */
/*    CTBSEND       send a request to the server                    */
/*                                                                  */
/* History:                                                         */
/*                                                                  */
/* Date    BTS#   Descrition                                        */
/* ======= ====== ================================================ */
/* Feb1795        Create                                            */
/* Oct3095 99999  Rewrite and add front end to the program          */
/*                                                                  */
/*******************************************************************/

DCL PLIXOPT CHAR(50) VAR INIT('NOSPIE, NOSTAE') STATIC EXTERNAL;

/*******************************************************************/
/* CLIENT LIBRARY PL/I COPY BOOK                                   */
/*******************************************************************/

        %INCLUDE CTPUBLIC;

/*-----------------------------------------------------------------*/
/* CICS BMS DEFINITIONS PL/I COPY BOOK                             */
/*-----------------------------------------------------------------*/

        %INCLUDE SYCTBA4;

/*-----------------------------------------------------------------*/
```

```
/* Standard CICS Attribute and Print Control Chararcter List        */
/*------------------------------------------------------------------*/

        %INCLUDE DFHBMSCA;

/*------------------------------------------------------------------*/
/* CICS Standard Attention Identifiers PL/I Copy Book               */
/*------------------------------------------------------------------*/

        %INCLUDE DFHAID;

/*------------------------------------------------------------------*/
/* CLIENT LIB ROUTINES DECLARATIONS                                 */
/*------------------------------------------------------------------*/

        DCL
            CSBCTXAL     ENTRY OPTIONS(INTER ASSEMBLER),
            CSBCTXDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBBIND      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCLOSE     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCMDAL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCMDDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCOMMA     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONAL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONDR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONPR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBCONNE     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBDESCR     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBDIAG      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBEXIT      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBFETCH     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBINIT      ENTRY OPTIONS(INTER ASSEMBLER),
            CTBPARAM     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBRESUL     ENTRY OPTIONS(INTER ASSEMBLER),
            CTBSEND      ENTRY OPTIONS(INTER ASSEMBLER);

/*------------------------------------------------------------------*/
/* BUILT IN FUNCTIONS DECLARATIONS                                  */
/*------------------------------------------------------------------*/
        DCL
            ADDR         BUILTIN,
            CSTG         BUILTIN,
            INDEX        BUILTIN,
            LENGTH       BUILTIN,
            STG          BUILTIN,
            SUBSTR       BUILTIN;
```

```
        DCL
            SYSPRINT STREAM OUTPUT PRINT ;

/*-------------------------------------------------------------------*/
/* WORK AREAS                                                        */
/*-------------------------------------------------------------------*/
        DCL
            01  CS_LIB_MISC_FIELDS,
                05  CSL_CMD_HANDLE        FIXED BIN(31) INIT(0),
                05  CSL_CON_HANDLE        FIXED BIN(31) INIT(0),
                05  CSL_CTX_HANDLE        FIXED BIN(31) INIT(0),
                05  CSL_NULL              FIXED BIN(31) INIT(0),
                05  CSL_RC                FIXED BIN(31);

        DCL
            01  PROPS_FIELDS,
                05  PF_SERVER             CHAR(30)      INIT(' '),
                05  PF_SERVER_SIZE        FIXED BIN(31) INIT(0),
                05  PF_USER               CHAR(08)      INIT(' '),
                05  PF_USER_SIZE          FIXED BIN(31) INIT(0),
                05  PF_PWD                CHAR(08)      INIT(' '),
                05  PF_PWD_SIZE           FIXED BIN(31) INIT(0),
                05  PF_TRAN               CHAR(08)      INIT(' '),
                05  PF_TRANL              FIXED BIN(31) INIT(0),
                05  PF_NETDRV             CHAR(08)      INIT(' '),
                05  PF_DRV_SIZE           FIXED BIN(31) INIT(0),
                05  PF_DEPT               CHAR(03)      INIT('D11'),
                05  PF_DEPT_SIZE          FIXED BIN(31) INIT(3),
                05  PF_MSGLIMIT           FIXED BIN(31),
                05  PF_STRLEN             FIXED BIN(31);
        DCL
            01  PARAM_FIELDS,
                05  PM_LEN                FIXED BIN(31),
                05  PM_PARAM1             FIXED BIN(31),
                05  PM_PARAM2             CHAR(03) VAR  INIT('   '),
                05  PM_NULLIND            FIXED BIN(15);
        DCL
            01  FETCH_FIELDS,
                05  FF_ROWS_READ          FIXED BIN(31),
                05  FF_ROW_NUM            FIXED BIN(31) INIT(0);
        DCL
            01  OUTPUT_ROW,
                05  OR_COL_FIRSTNME       PIC'(12)X',
                05  SPACE1                CHAR(01)      INIT(' '),
                05  OR_COL_LASTNAME       PIC'(15)X',
```

```
        05  SPACE2                 CHAR(01)       INIT(' '),
        05  OR_COL_EDUCLVL         PIC'ZZ9',
        05  SPACE3                 CHAR(08)       INIT(' '),
        05  OR_COL_JOBCODE         PIC'ZZ9V.',
        05  SPACE4                 CHAR(06)       INIT(' '),
        05  OR_COL_SALARY          PIC'ZZ,ZZZV.99';
DCL
    01  OUTPUT_ROW_STR             CHAR(59)
                                    DEFINED OUTPUT_ROW;
DCL
    01  OUTPUT_ROW2,
        05  OR2_PAREN               CHAR(01)       INIT('('),
        05  OR2_COL_RET1            PIC'99999',
        05  OR2_MESSAGE             CHAR(17)
            INIT(' row(s) affected)');
DCL
    01  OUTPUT_ROW_STR2            CHAR(23)
                                    DEFINED OUTPUT_ROW2;
DCL
    01  COLUMN_FIELDS,
        05  CF_COL_FIRSTNME        CHAR(12) VAR,
        05  CF_COL_LASTNAME        CHAR(15) VAR,
        05  CF_COL_EDUCLVL         FIXED BIN(15) INIT(0),
        05  CF_COL_JOBCODE         FIXED DEC(5,2),
        05  CF_COL_SALARY          FIXED DEC(8,2),
        05  CF_COL_LEN             FIXED BIN(31),
        05  CF_COL_NULL            FIXED BIN(31) INIT(0),
        05  CF_COL_NUMBER          FIXED BIN(31) INIT(1),
        05  CF_COL_INDICATOR       FIXED BIN(15) INIT(0);
DCL
    01  CMD_FIELDS                 STATIC,
        05  CF_CMD                 CHAR(04)       INIT('SYR2');
DCL
    01  RESULTS_FIELDS,
        05  RF_TYPE                FIXED BIN(31);

DCL
    01  DATAFMT,
        05  DF_NAME                CHAR(132),
        05  DF_NAMELEN             FIXED BIN(31),
        05  DF_DATATYPE            FIXED BIN(31),
        05  DF_FORMAT              FIXED BIN(31),
        05  DF_MAXLENGTH           FIXED BIN(31),
        05  DF_SCALE               FIXED BIN(31),
        05  DF_PRECISION           FIXED BIN(31),
        05  DF_STATUS              FIXED BIN(31),
```

```
        05  DF_COUNT              FIXED BIN(31),
        05  DF_USERTYPE           FIXED BIN(31),
        05  DF_LOCALE             CHAR(68);

DCL
    01 DISP_MSG,
       05 TEST_CASE              CHAR(09) INIT('SYCTSAR4 '),
       05 MSG,
          10 SAMP_LIT            CHAR(05) INIT('rc = '),
          10 SAMP_RC             PIC'99',
          10 REST_LIT            CHAR(15)
                                   INIT('  Result Type: '),
          10 REST_TYPE           PIC'9999',
          10 FILLER              CHAR(03) INIT(' '),
          10 MSGSTR              CHAR(40) INIT(' ');

DCL
    BAD_INPUT                    BIT(01)       INIT('0'B),
    BLANK                        CHAR(01)      INIT(' '),
    BLANK_13                     CHAR(13)      INIT(' '),
    FALSE                        BIT(01)       INIT('0'B),
    I1                           FIXED BIN(15) INIT(0),
    MAX_SCREEN_ROWS              FIXED BIN(15) INIT(07 ),
    MSG_TEXT_1                   CHAR(79)      INIT(' '),
    MSG_TEXT_2                   CHAR(79)      INIT(' '),
    PAGE_CNT                     FIXED BIN(31) INIT(0),
    OUTLEN                       FIXED BIN(31) INIT(0),
    PARM_CNT                     FIXED BIN(31),
    NETDRIVER                    FIXED BIN(31) INIT(9999),
    STRLEN                       FIXED BIN(31) INIT(0),
    TMP_TIME                     CHAR(08)      INIT(' '),
    TMP_DATE                     CHAR(08)      INIT(' '),
    TRUE                         BIT(01)       INIT('1'B),
    UTIME                        FIXED DEC(15) INIT(0);

DCL
    DIAG_MSGS_INITIALIZED        BIT(1)   INIT('0'B),
    ENTER_DATA_SW                BIT(1)   INIT('0'B),
    NO_ERRORS_SW                 BIT(1)   INIT('0'B),
    NO_MORE_RESULTS              BIT(1)   INIT('0'B),
    NO_MORE_ROWS                 BIT(1)   INIT('0'B),
    PRINT_ONCE                   BIT(1)   INIT('1'B);

DCL
    01  QUERY_FIELDS,
        05  QF_LEN              FIXED BIN(15) INIT(1),
```

```
            05  QF_MAXLEN              FIXED BIN(15) INIT(1),
            05  QF_ANSWER              CHAR(01)      INIT(' ');

    DCL
        01  CANCELED_FIELDS,
            05  CICS_RESPONSE          FIXED BIN(31);

    DCL
        01  DIAG_FIELDS,
            05  DG_MSGNO               FIXED BIN(31) INIT(1),
            05  DG_NUM_OF_MSGS         FIXED BIN(31) INIT(0);

/*-------------------------------------------------------------------*/
/* Client Message Structure                                          */
/*-------------------------------------------------------------------*/

    DCL
        01 CLIENT_MSG,
           05 CM_SEVERITY              FIXED BIN(31),
           05 CM_MSGNO                 FIXED BIN(31),
           05 CM_TEXT                  CHAR(256),
           05 CM_TEXT_LEN              FIXED BIN(31),
           05 CM_OS_MSGNO              FIXED BIN(31),
           05 CM_OS_MSGTXT             CHAR(256),
           05 CM_OS_MSGTEXT_LEN        FIXED BIN(31),
           05 CM_STATUS                FIXED BIN(31);

    DCL
        01 DISP_CLIENT_MSG_1,
           05 CM_SEVERITY_HDR          CHAR(13)
                                        INIT('  Severity:  '),
           05 CM_SEVERITY_DATA         PIC'ZZZ9',
           05 CM_STATUS_HDR            CHAR(12)
                                        INIT(',  Status:  '),
           05 CM_STATUS_DATA           PIC'ZZZ9' ;

    DCL
        01 DISP_CLIENT_MSG_2,
           05 CM_OC_MSGNO_HDR          CHAR(13)
                                        INIT('  OC MsgNo:  '),
           05 CM_OC_MSGNO_DATA         PIC'ZZZZZZZ9' ;

    DCL
        01 DISP_CLIENT_MSG_3,
           05 CM_OC_MSG_HDR            CHAR(13)
                                        INIT('  OC MsgTx:  '),
```

```
            05 CM_OC_MSG_DATA          CHAR(66);

        DCL
            01 DISP_CLIENT_MSG_3A,
               05 CM_OC_MSG_DATA_1     CHAR(66),
               05 CM_OC_MSG_DATA_2     CHAR(66),
               05 CM_OC_MSG_DATA_3     CHAR(66),
               05 CM_OC_MSG_DATA_4     CHAR(58);

        DCL
            01 DISP_CLIENT_MSG_3B,
               05 FILLER               CHAR(13) INIT(' '),
               05 CM_OC_MSG_DATA_X     CHAR(66);

        DCL
            01 DISP_CLIENT_MSG_4,
               05 CM_OS_MSG_HDR        CHAR(13)
                                        INIT('  OS MsgNo:  '),
               05 CM_OS_MSGNO_DATA     PIC'ZZZZZZZ9' ;

        DCL
            01 DISP_CLIENT_MSG_5,
               05 CM_OS_MSG_HDR        CHAR(13)
                                        INIT('  OS MsgTx:  '),
               05 CM_OS_MSG_DATA       CHAR(66);

/*-------------------------------------------------------------------*/
/* Server Message Structure                                          */
/*-------------------------------------------------------------------*/

        DCL
            01 SERVER_MSG,
               05 SM_MSGNO             FIXED BIN(31),
               05 SM_STATE             FIXED BIN(31),
               05 SM_SEV               FIXED BIN(31),
               05 SM_TEXT              CHAR(256),
               05 SM_TEXT_LEN          FIXED BIN(31),
               05 SM_SVRNAME           CHAR(256),
               05 SM_SVRNAME_LEN       FIXED BIN(31),
               05 SM_PROC              CHAR(256),
               05 SM_PROC_LEN          FIXED BIN(31),
               05 SM_LINE              FIXED BIN(31),
               05 SM_STATUS            FIXED BIN(31);

        DCL
            01 DISP_SERVER_MSG_1,
```

```
            05 SM_MSG_NO_HDR          CHAR(13)
                                       INIT('  Message#:  '),
            05 SM_MSG_NO_DATA         PIC'ZZZZZZZ9',
            05 SM_SEVERITY_HDR        CHAR(14)
                                       INIT(',  Severity:  '),
            05 SM_SEVERITY_DATA       PIC'ZZZ9',
            05 SM_STATE_HDR           CHAR(14)
                                       INIT(',  State No:  '),
            05 SM_STATE_DATA          PIC'ZZZ9' ;

        DCL
            01 DISP_SERVER_MSG_2,
            05 SM_LINE_NO_HDR         CHAR(13)
                                       INIT('  Line  No:  '),
            05 SM_LINE_NO_DATA        PIC'ZZZ9',
            05 SM_STATUS_HDR          CHAR(14)
                                       INIT(',  Status  :  '),
            05 SM_STATUS_DATA         PIC'ZZZ9' ;

        DCL
            01 DISP_SERVER_MSG_3,
            05 SM_SVRNAME_HDR         CHAR(13)
                                       INIT('  Serv Nam:  '),
            05 SM_SVRNAME_DATA        CHAR(66) ;

        DCL
            01 DISP_SERVER_MSG_4,
            05 SM_PROC_ID_HDR         CHAR(13)
                                       INIT('  Proc  ID:  '),
            05 SM_PROC_ID_DATA        CHAR(66);

        DCL
            01 DISP_SERVER_MSG_5,
            05 SM_MSG_HDR             CHAR(13)
                                       INIT('  Message :  '),
            05 SM_MSG_DATA            CHAR(66);

        DCL
            01 DISP_SERVER_MSG_5X,
            05 FILLER                 CHAR(13) INIT(' '),
            05 SM_MSG_DATA_X          CHAR(66);

/*-------------------------------------------------------------------*/
/* CICS Condition Handler                                            */
/*-------------------------------------------------------------------*/
```

```
        EXEC CICS HANDLE CONDITION MAPFAIL(NO_INPUT)
                                   ERROR(ERRORS) ;

 /*-------------------------------------------------------------------*/
 /* CICS Aid Handler                                                  */
 /*-------------------------------------------------------------------*/

        EXEC CICS HANDLE AID ANYKEY(NO_INPUT)
                             CLEAR(GETOUT) ;

 /*-------------------------------------------------------------------*/
 /* program initialization                                            */
 /*-------------------------------------------------------------------*/

        DIAG_MSGS_INITIALIZED = TRUE ;
        MSG_TEXT_2            = 'Press Clear To Exit';
        NO_ERRORS_SW          = TRUE ;
        PAGE_CNT              = PAGE_CNT + 1;
        SERVERL               = -1 ;

        DO I1 = 1 TO 13 ;
          RSLTNO( I1 ) = BLANK ;
        END ;

        CALL GET_SYSTEM_TIME ;

 GET_INPUT_AGAIN:

          CALL DISPLAY_INITIAL_SCREEN ;
          CALL GET_INPUT_DATA ;

/*-------------------------------------------------------------------*/
 /* allocate a context structure                                      */
 /*-------------------------------------------------------------------*/

        CALL CSBCTXAL( CS_VERSION_46,
                       CSL_RC,
                       CSL_CTX_HANDLE );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            NO_ERRORS_SW = FALSE ;
            MSGSTR       = 'CSCTXALLOC failed';
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;
```

```
/*--------------------------------------------------------------------*/
/* initialize the Client-Library                                      */
/*--------------------------------------------------------------------*/

        CALL CTBINIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_VERSION_46 );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            NO_ERRORS_SW = FALSE ;
            MSGSTR       = 'CTBINIT failed';
            CALL ERROR_OUT;
            CALL ALL_DONE;
        END;

        CALL PROCESS_INPUT ;

        CALL QUIT_CLIENT_LIBRARY ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to get system time                                      */
/*                                                                    */
/*--------------------------------------------------------------------*/
GET_SYSTEM_TIME: PROC ;

        EXEC CICS ASKTIME ABSTIME(UTIME);

        EXEC CICS FORMATTIME
                    ABSTIME(UTIME)
                    DATESEP('/')
                    MMDDYY(TMP_DATE)
                    TIME(TMP_TIME)
                    TIMESEP ;

END GET_SYSTEM_TIME ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to get system time                                      */
/*                                                                    */
/*--------------------------------------------------------------------*/
DISPLAY_INITIAL_SCREEN: PROC ;
```

```
        SDATEO    = TMP_DATE ;
        STIMEO    = TMP_TIME ;
        MSG1O     = MSG_TEXT_1 ;
        PROGNMO   = 'SYCTSAR4' ;
        MSG1O     = MSG_TEXT_1 ;
        MSG2O     = MSG_TEXT_2 ;
        SPAGEO    = '0001' ;

        EXEC CICS SEND MAP('A4PANEL')
                       MAPSET('SYCTBA4')
                       CURSOR
                       FRSET
                       ERASE
                       FREEKB ;

END DISPLAY_INITIAL_SCREEN ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to get input data                                       */
/*                                                                    */
/*--------------------------------------------------------------------*/
GET_INPUT_DATA: PROC ;

        EXEC CICS RECEIVE MAP('A4PANEL')
                          MAPSET('SYCTBA4')
                          ASIS ;

        IF SERVERL = 0 THEN
        DO ;
          IF PF_SERVER = BLANK THEN
          DO ;
            SERVERL        = -1 ;        /* set the cursor position */
            MSG_TEXT_1     = 'Please Enter Server Name' ;
            ENTER_DATA_SW  = TRUE ;
          END ;
        END ;
        ELSE DO ;
          PF_SERVER      = SERVERI ;
          PF_SERVER_SIZE = SERVERL ;
        END ;

        IF USERL = 0 THEN
        DO ;
          IF PF_USER = BLANK THEN
          DO ;
```

```
            USERL          = -1 ;        /* set the cursor position */
            MSG_TEXT_1     = 'Please Enter User-ID' ;
            ENTER_DATA_SW  = TRUE ;
          END ;
        END ;
        ELSE DO ;
          PF_USER      = USERI ;
          PF_USER_SIZE = USERL ;
        END ;

        IF PSWDL ^= 0 THEN
        DO ;
          PF_PWD      = PSWDI;
          PF_PWD_SIZE = PSWDL ;
        END ;

        IF TRANL ^= 0 THEN
        DO ;
          PF_TRAN     = TRANI;
          PF_TRANL = TRANL ;
        END ;

        IF NETDRVL ^= 0 THEN
        DO ;
          PF_NETDRV   = NETDRVI ;
          PF_DRV_SIZE = NETDRVL ;
        END ;

        IF ENTER_DATA_SW = TRUE THEN
          DO ;
            ENTER_DATA_SW = FALSE ;
            CALL DISPLAY_INITIAL_SCREEN ;
            MSG_TEXT_1    = BLANK ;
            CALL GET_INPUT_DATA ;
          END ;

END GET_INPUT_DATA ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process input data                                  */
/*                                                                   */
/*-------------------------------------------------------------------*/
PROCESS_INPUT: PROC ;

/*-------------------------------------------------------------*/
```

```
/* allocate a connection to the server                          */
/*--------------------------------------------------------------*/

        CSL_CON_HANDLE = 0 ;

        CALL CTBCONAL( CSL_CTX_HANDLE,
                       CSL_RC,
                       CSL_CON_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCONALLOC failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*--------------------------------------------------------------*/
/* alter properties of the connection for user-id               */
/*--------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_USERNAME,
                       PF_USER,
                       PF_USER_SIZE,
                       CS_FALSE,
                       OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCONPROPS for user-id failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*--------------------------------------------------------------*/
/* alter properties of the connection for password              */
/*--------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                       CSL_RC,
                       CS_SET,
                       CS_PASSWORD,
```

```
                        PF_PWD,
                        PF_PWD_SIZE,
                        CS_FALSE,
                        OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCONPROPS for password failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for transaction         */
/*-------------------------------------------------------------*/

        CALL CTBCONPR( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_SET,
                        CS_TRANSACTION_NAME,
                        PF_TRAN,
                        PF_TRANL,
                        CS_FALSE,
                        OUTLEN ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBCONPROPS for transaction failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* alter properties of the connection for Network driver      */
/*-------------------------------------------------------------*/

        SELECT;
          WHEN (PF_NETDRV = '        ')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'LU62' | PF_NETDRV = 'lu62')
                NETDRIVER = CS_LU62 ;
          WHEN (PF_NETDRV = 'IBMTCPIP' | PF_NETDRV = 'ibmtcpip')
                NETDRIVER = CS_TCPIP ;
          WHEN (PF_NETDRV = 'INTERLIN' | PF_NETDRV = 'interlin')
```

```
            NETDRIVER = CS_INTERLINK ;
        WHEN (PF_NETDRV = 'CPIC' | PF_NETDRV = 'cpic')
            NETDRIVER = CS_NCPIC ;
        OTHERWISE
          DO;
            MSGSTR = 'Invalid Network driver entered';
            NO_ERRORS_SW = FALSE ;
            CALL ERROR_OUT;
            CALL ALL_DONE ;
          END;
      END;

      CALL CTBCONPR( CSL_CON_HANDLE,
                     CSL_RC,
                     CS_SET,
                     CS_NET_DRIVER,
                     NETDRIVER,
                     CS_UNUSED,
                     CS_FALSE,
                     OUTLEN ) ;

      IF CSL_RC ^= CS_SUCCEED THEN
      DO ;
        MSGSTR = 'CTBCONPROPS for Network driver failed' ;
        NO_ERRORS_SW = FALSE ;
        CALL ERROR_OUT;
        CALL ALL_DONE ;
      END ;

/*-------------------------------------------------------------*/
/* setup retrieval of All Messages                             */
/*-------------------------------------------------------------*/

      CALL CTBDIAG( CSL_CON_HANDLE,
                    CSL_RC,
                    CS_UNUSED,
                    CS_INIT,
                    CS_ALLMSG_TYPE,
                    CS_UNUSED,
                    CS_UNUSED ) ;

      IF CSL_RC ^= CS_SUCCEED THEN
      DO ;
        NO_ERRORS_SW = FALSE ;
        MSGSTR       = 'CTBDIAG CS_INIT failed' ;
        CALL ERROR_OUT;
```

```
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* set the upper limit of number of messages                   */
/*-------------------------------------------------------------*/

        PF_MSGLIMIT = 5 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_MSGLIMIT,
                      CS_ALLMSG_TYPE,
                      CS_UNUSED,
                      PF_MSGLIMIT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBDIAG CS_MSGLIMIT failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* open connection to the server or CICS region                */
/*-------------------------------------------------------------*/

        CALL CTBCONNE( CSL_CON_HANDLE,
                       CSL_RC,
                       PF_SERVER,
                       PF_SERVER_SIZE,
                       CS_FALSE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCONNECT failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* invokes SEND_COMMAND routine                                */
/*-------------------------------------------------------------*/
```

```
        IF NO_ERRORS_SW
          THEN
            CALL SEND_PARAM ;

 /*-------------------------------------------------------------*/
 /* process the results of the command                          */
 /*-------------------------------------------------------------*/

        IF NO_ERRORS_SW THEN
        DO ;
          DO WHILE( ^NO_MORE_RESULTS ) ;
            CALL PROCESS_RESULTS ;
          END ;
          CALL CLOSE_CONNECTION ;
        END ;

 END PROCESS_INPUT ;

 /*-------------------------------------------------------------------*/
 /*                                                                   */
 /* Subroutine to allocate, send, and process commands                */
 /*                                                                   */
 /*-------------------------------------------------------------------*/
 SEND_PARAM: PROC ;

/*-------------------------------------------------------------*/
 /* allocate a command handle                                   */
 /*-------------------------------------------------------------*/

        CALL CTBCMDAL( CSL_CON_HANDLE,
                       CSL_RC,
                       CSL_CMD_HANDLE ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCMDALLOC failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

 /*-------------------------------------------------------------*/
 /* prepare the command (an RPC request)                        */
 /*-------------------------------------------------------------*/

        PF_STRLEN = STG(CF_CMD);
```

```
        CALL CTBCOMMA( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_RPC_CMD,
                       CF_CMD,
                       PF_STRLEN,
                       CS_UNUSED );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       = 'CTBCOMMAND failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/*                                                             */
/* setup a return parameter for NUM_OF_ROWS                    */
/*                                                             */
/* describe the first parameter (NUM_OF_ROWS)                  */
/*                                                             */
/*-------------------------------------------------------------*/

        DF_NAME         = '@parm1';
        DF_NAMELEN      = 6;
        DF_DATATYPE     = CS_INT_TYPE;
        DF_FORMAT       = CS_FMT_UNUSED;
        DF_MAXLENGTH    = CS_UNUSED;
        DF_STATUS       = CS_RETURN;
        DF_USERTYPE     = CS_UNUSED;

        PM_LEN          = STG(PM_PARAM1);
        PM_PARAM1       = 0;                    /* NUM_OF_ROWS */
        PM_NULLIND      = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
                       CSL_RC,
                       DATAFMT,
                       PM_PARAM1,
                       PM_LEN,
                       PM_NULLIND );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
```

```
          MSGSTR       =
            'CTBPARAM CS_INT_TYPE parm1 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/*                                                             */
/* describe the second parameter (DEPTNO)                      */
/*                                                             */
/*-------------------------------------------------------------*/

        DF_NAME          = '@parm2';
        DF_NAMELEN       = 6;
        DF_DATATYPE      = CS_VARCHAR_TYPE;
        DF_FORMAT        = CS_FMT_UNUSED;
        DF_MAXLENGTH     = CS_UNUSED;
        DF_STATUS        = CS_INPUTVALUE;
        DF_USERTYPE      = CS_UNUSED;

        PM_PARAM2        = PF_DEPT;                 /* DEPTNO */
        PM_LEN           = PF_DEPT_SIZE ;
        PM_NULLIND       = 0;

        CALL CTBPARAM( CSL_CMD_HANDLE,
                       CSL_RC,
                       DATAFMT,
                       PM_PARAM2,
                       PM_LEN,
                       PM_NULLIND );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBPARAM CS_VARCHAR_TYPE parm2 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* send the command                                            */
/*-------------------------------------------------------------*/

        CALL CTBSEND( CSL_CMD_HANDLE,
                      CSL_RC ) ;
```

```
         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           MSGSTR       = 'CTBSEND failed' ;
           NO_ERRORS_SW = FALSE ;
           CALL ERROR_OUT;
           CALL ALL_DONE ;
         END ;

 END SEND_PARAM ;

 /*-------------------------------------------------------------------*/
 /*                                                                   */
 /* Subroutine to process the result                                  */
 /*                                                                   */
 /*-------------------------------------------------------------------*/
 PROCESS_RESULTS: PROC ;

 /*-----------------------------------------------------------*/
 /* set up the results data                                   */
 /*-----------------------------------------------------------*/

         CALL CTBRESUL( CSL_CMD_HANDLE,
                        CSL_RC,
                        RF_TYPE ) ;

 /*-----------------------------------------------------------*/
 /* determine the outcome of the comand execution             */
 /*-----------------------------------------------------------*/

         SELECT( CSL_RC ) ;

           WHEN( CS_SUCCEED )
           DO ;

 /*-------------------------------------------------------------*/
 /* determine the type of result returned by the current request */
 /*-------------------------------------------------------------*/

             SELECT( RF_TYPE ) ;

/*-----------------------------------------------------------*/
 /* process row results                                       */
 /*-----------------------------------------------------------*/

               WHEN( CS_ROW_RESULT )
```

```
              DO ;
                CALL RESULT_ROW_PROCESSING ;
                DO WHILE( ^NO_MORE_ROWS ) ;
                  CALL FETCH_ROW_PROCESSING ;
                END ;
              END ;

 /*-------------------------------------------------------------*/
 /* process parameter results --- there should be no parameter */
 /* to process                                                  */
 /*-------------------------------------------------------------*/

              WHEN( CS_PARAM_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;
                CALL RESULT_PARAM_PROCESSING ;
                CALL FETCH_PARAM_PROCESSING ;
              END ;

 /*-------------------------------------------------------------*/
 /* process status results --- the stored procedure status     */
 /* result will not be processed in this example               */
 /*-------------------------------------------------------------*/

              WHEN( CS_STATUS_RESULT )
              DO ;
                NO_MORE_ROWS = FALSE ;

                CALL CTBFETCH( CSL_CMD_HANDLE,
                               CSL_RC,
                               CS_UNUSED,             /* type */
                               CS_UNUSED,           /* offset */
                               CS_UNUSED,           /* option */
                               FF_ROWS_READ );

                IF CSL_RC ^= CS_SUCCEED THEN
                DO ;
                  MSGSTR       = 'CTBFETCH status failed' ;
                  NO_ERRORS_SW = FALSE ;
                  CALL ERROR_OUT;
                  CALL ALL_DONE ;
                END ;
              END ;

/*-------------------------------------------------------------*/
 /* print an error message if the server encountered an error  */
```

```
/* while executing the request                                    */
/*----------------------------------------------------------------*/

              WHEN( CS_CMD_FAIL )
              DO ;
                NO_ERRORS_SW = FALSE ;
                MSGSTR       =
                  'CTBRESUL failed with CS_CMD_FAIL restype' ;
                CALL ERROR_OUT ;
              END ;

/*----------------------------------------------------------------*/
/* print a message for successful commands that returned no       */
/* data( optional )                                               */
/*----------------------------------------------------------------*/

              WHEN( CS_CMD_SUCCEED )
              DO ;
                MSGSTR =
                  'CTBRESUL returned CS_CMD_SUCCEED restype' ;
              END ;

/*----------------------------------------------------------------*/
/* print a message for requests that have been processed          */
/* successfully( optional )                                       */
/*----------------------------------------------------------------*/

              WHEN( CS_CMD_DONE )
              DO ;
                MSGSTR =
                  'CTBRESUL returned CS_CMD_DONE restype' ;
              END ;

              OTHERWISE
              DO ;
                NO_MORE_RESULTS = TRUE ;
                NO_ERRORS_SW    = FALSE ;
                MSGSTR          =
                  'CTBRESUL returned UNKNOWN restype' ;
                CALL ERROR_OUT ;

              END ;
            END ; /* end of SELECT( RF_TYPE ) */
          END ;

/*----------------------------------------------------------------*/
```

```
/* print an error message if the CTBRESULTS call failed       */
/*-------------------------------------------------------------*/

          WHEN( CS_FAIL )
          DO ;
            NO_MORE_RESULTS = TRUE ;
            NO_ERRORS_SW    = FALSE ;
            MSGSTR          =
              'CTBRESULTS failed with CS_FAIL ret_cd' ;
            CALL ERROR_OUT ;
          END ;

/*-------------------------------------------------------------*/
/* drop out of the results loop if no more result sets are    */
/* available for processing or if the results were cancelled  */
/*-------------------------------------------------------------*/

          WHEN( CS_END_RESULTS )
          DO ;
            NO_MORE_RESULTS = TRUE ;
          END ;

          WHEN( CS_CANCELLED )
          DO ;
            NO_MORE_RESULTS = TRUE ;
          END ;

          OTHERWISE
          DO ;
            NO_MORE_RESULTS = TRUE ;
            NO_ERRORS_SW    = FALSE ;
            MSGSTR          =
               'CTBRESUL failed with unknown ret_cd' ;
            CALL ERROR_OUT ;
          END ;
        END ; /* end of SELECT( CSL_RC ) */

        RF_TYPE = 0 ;

END PROCESS_RESULTS ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to process result rows                                 */
/*                                                                   */
/*-------------------------------------------------------------------*/
```

```
 RESULT_ROW_PROCESSING: PROC ;

         CALL BIND_ROW_PROCESSING ;

 END RESULT_ROW_PROCESSING ;

/*--------------------------------------------------------------------*/
 /*                                                                    */
 /* Subroutine to bind each data                                       */
 /*                                                                    */
 /*--------------------------------------------------------------------*/
 BIND_ROW_PROCESSING: PROC ;

 /*-------------------------------------------------------------*/
 /* We need to bind the data to program variables. We don't     */
 /* care about the indicator variable so we'll pass NULL for    */
 /* that PARAMeter in OC_BIND().                                */
 /*-------------------------------------------------------------*/

 /*-------------------------------------------------------------*/
 /* bind the first column, FIRSTNME defined as VARCHAR(12)      */
 /*-------------------------------------------------------------*/

         DF_DATATYPE   = CS_VARCHAR_TYPE;
         DF_FORMAT     = CS_FMT_UNUSED;
         DF_MAXLENGTH  = STG(CF_COL_FIRSTNME) - 2;
         DF_COUNT      = 1;                  /* rows per fetch */
         CF_COL_NUMBER = 1;           /* bind the first column */

         CALL CTBBIND( CSL_CMD_HANDLE,
                       CSL_RC,
                       CF_COL_NUMBER,
                       DATAFMT,
                       CF_COL_FIRSTNME,
                       CF_COL_LEN,
                       CS_PARAM_NOTNULL,
                       CF_COL_INDICATOR,
                       CS_PARAM_NULL );

         IF CSL_RC ^= CS_SUCCEED THEN
         DO ;
           NO_ERRORS_SW = FALSE ;
           MSGSTR       =
             'CTBBIND CS_VARCHAR_TYPE column 1 failed' ;
           CALL ERROR_OUT;
           CALL ALL_DONE;
```

```
        END ;

/*-------------------------------------------------------------*/
/* bind the second column, LASTNAME defined as VARCHAR(15)     */
/*-------------------------------------------------------------*/

        DF_MAXLENGTH  = STG(CF_COL_LASTNAME) - 2;
        CF_COL_NUMBER = 2;          /* bind the second column */

        CALL CTBBIND( CSL_CMD_HANDLE,
                      CSL_RC,
                      CF_COL_NUMBER,
                      DATAFMT,
                      CF_COL_LASTNAME,
                      CF_COL_LEN,
                      CS_PARAM_NOTNULL,
                      CF_COL_INDICATOR,
                      CS_PARAM_NULL );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBBIND CS_VARCHAR_TYPE column 2 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

/*-------------------------------------------------------------*/
/* bind the third column, EDUCLVL defined as SMALLINT          */
/*-------------------------------------------------------------*/

        DF_DATATYPE   = CS_SMALLINT_TYPE;
        DF_MAXLENGTH  = CS_UNUSED;
        CF_COL_NUMBER = 3;           /* bind the third column */

        CALL CTBBIND( CSL_CMD_HANDLE,
                              CSL_RC,
                              CF_COL_NUMBER,
                              DATAFMT,
                              CF_COL_EDUCLVL,
                              CF_COL_LEN,
                              CS_PARAM_NOTNULL,
                              CF_COL_INDICATOR,
                              CS_PARAM_NULL );
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBBIND CS_SMALLINT_TYPE column 3 failed' ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

/*-------------------------------------------------------------*/
/* bind the fourth column, JOBCODE as DECIMAL(3,0). It will    */
/* convert from float8 or money.                               */
/*-------------------------------------------------------------*/

        DF_DATATYPE   = CS_PACKED370_TYPE;
        DF_SCALE      = CS_SRC_VALUE;
        DF_PRECISION  = CS_SRC_VALUE;
        DF_MAXLENGTH  = STG(CF_COL_JOBCODE);
        CF_COL_NUMBER = 4;          /* bind the fourth column */
        CF_COL_JOBCODE = 1;
        CALL CTBBIND( CSL_CMD_HANDLE,
                      CSL_RC,
                      CF_COL_NUMBER,
                      DATAFMT,
                      CF_COL_JOBCODE,
                      CF_COL_LEN,
                      CS_PARAM_NOTNULL,
                      CF_COL_INDICATOR,
                      CS_PARAM_NULL );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBBIND CS_PACKED370_TYPE column 4 fail' ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

/*-------------------------------------------------------------*/
/* bind the fifth column, SALARY as DECIMAL(8,2). It will      */
/* convert from money.                                         */
/*-------------------------------------------------------------*/

        DF_MAXLENGTH  = STG(CF_COL_SALARY);
        CF_COL_NUMBER = 5;           /* bind the fifth column */
```

```
        CF_COL_SALARY = 0;
        CALL CTBBIND( CSL_CMD_HANDLE,
                      CSL_RC,
                      CF_COL_NUMBER,
                      DATAFMT,
                      CF_COL_SALARY,
                      CF_COL_LEN,
                      CS_PARAM_NOTNULL,
                      CF_COL_INDICATOR,
                      CS_PARAM_NULL );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          NO_ERRORS_SW = FALSE ;
          MSGSTR       =
            'CTBBIND CS_PACKED370_TYPE column 5 fail' ;
          CALL ERROR_OUT;
          CALL ALL_DONE;
        END ;

 END BIND_ROW_PROCESSING ;

/*--------------------------------------------------------------------*/
 /*                                                                   */
 /* Subroutine to fetch row processing                                */
 /*                                                                   */
 /*--------------------------------------------------------------------*/
 FETCH_ROW_PROCESSING: PROC ;

        CALL CTBFETCH( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_UNUSED,       /* type   */
                       CS_UNUSED,       /* offset */
                       CS_UNUSED,       /* option */
                       FF_ROWS_READ ) ;

        SELECT( CSL_RC ) ;

          WHEN( CS_SUCCEED )
          DO ;
            NO_MORE_ROWS = FALSE ;
            IF FF_ROW_NUM = 0 THEN
            DO ;
              FF_ROW_NUM          = FF_ROW_NUM + 1;
              RSLTNO(FF_ROW_NUM)  = 'FirstName    '     ||
                                    'LastName        ' ||
```

```
                                'EducLvl     '       ||
                                'JobCode    '        ||
                                'Salary'              ;
                FF_ROW_NUM          = FF_ROW_NUM + 1;
                RSLTNO(FF_ROW_NUM)  = '============ '     ||
                                '===============  ' ||
                                '=======     '       ||
                                '=======    '        ||
                                '=========='         ;
              END ;

              FF_ROW_NUM            = FF_ROW_NUM + 1;

              IF FF_ROW_NUM > MAX_SCREEN_ROWS THEN
              DO ;
                MSG_TEXT_1 = 'Please press return to continue!' ;
                MSG_TEXT_2 = BLANK ;
                CALL DISP_DATA ;

                DO FF_ROW_NUM = 1 TO MAX_SCREEN_ROWS ;
                  RSLTNO( MAX_SCREEN_ROWS ) = BLANK ;
                END ;

                PAGE_CNT            = PAGE_CNT + 1 ;
                FF_ROW_NUM          = 1;
                RSLTNO(FF_ROW_NUM)  = 'FirstName     '     ||
                                'LastName         ' ||
                                'EducLvl     '       ||
                                'JobCode    '        ||
                                'Salary'              ;
                FF_ROW_NUM          = FF_ROW_NUM + 1;
                RSLTNO(FF_ROW_NUM)  = '============ '     ||
                                '===============  ' ||
                                '=======     '       ||
                                '=======    '        ||
                                '=========='         ;
                FF_ROW_NUM          = FF_ROW_NUM + 1;
              END ;

/*-------------------------------------------------------------*/
/* display results                                             */
/*-------------------------------------------------------------*/

              OR_COL_FIRSTNME = CF_COL_FIRSTNME;
              OR_COL_LASTNAME = CF_COL_LASTNAME;
              OR_COL_EDUCLVL  = CF_COL_EDUCLVL;
```

```
  OR_COL_JOBCODE  = CF_COL_JOBCODE;
  OR_COL_SALARY   = CF_COL_SALARY;

  RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR;

END ; /* end of WHEN( CS_SUCCEED ) */

WHEN( CS_END_DATA )
DO ;
  NO_MORE_ROWS = TRUE ;
  MSG_TEXT_1   = 'All rows processing completed!' ;
  MSG_TEXT_2   = 'Press Clear To Exit';
  CALL DISP_DATA ;
END ; /* end of WHEN( CS_END_DATA ) */

WHEN( CS_FAIL )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
      'CTBFETCH returned CS_FAIL ret_cd' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_FAIL ) */

WHEN( CS_ROW_FAIL )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
      'CTBFETCH returned CS_ROW_FAIL ret_cd' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_ROW_FAIL ) */

WHEN( CS_CANCELLED )
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       = 'CTBFETCH returned CS_CANCELLED ret_cd' ;
  CALL ERROR_OUT;
END ; /* end of WHEN( CS_CANCELLED ) */

OTHERWISE
DO ;
  NO_MORE_ROWS = TRUE ;
  NO_ERRORS_SW = FALSE ;
  MSGSTR       =
```

```
                'CTBFETCH returned Unknown ret_cd' ;
            CALL ERROR_OUT;
          END ; /* end of OTHERWISE */

        END ; /* end of SELECT( CSL_RC ) */

END FETCH_ROW_PROCESSING ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to describe the returned parameters                     */
/*                                                                    */
/*--------------------------------------------------------------------*/

RESULT_PARAM_PROCESSING: PROC;

        I1 = 1;
        CALL CTBDESCR( CSL_CMD_HANDLE,
                       CSL_RC,
                       I1,
                       DATAFMT ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBDESCR failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

/*-------------------------------------------------------------*/
/* bind the return parameter, NUM_OF_ROWS defined as INTEGER  */
/*-------------------------------------------------------------*/

        DF_DATATYPE   = CS_INT_TYPE;
        DF_FORMAT     = CS_FMT_UNUSED;
        DF_MAXLENGTH  = CS_UNUSED;
        DF_COUNT      = 1;                  /* rows per fetch */
        CF_COL_NUMBER = 1;           /* bind the first column */

        CALL CTBBIND( CSL_CMD_HANDLE,
                      CSL_RC,
                      CF_COL_NUMBER,
                      DATAFMT,
                      PM_PARAM1,
                      CF_COL_LEN,
```

```
                        CS_PARAM_NOTNULL,
                        CF_COL_INDICATOR,
                        CS_PARAM_NULL );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBBIND return parameter failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

 END RESULT_PARAM_PROCESSING ;

 /*--------------------------------------------------------------------*/
 /*                                                                    */
 /* Subroutine to fetch return parameter                               */
 /*                                                                    */
 /*--------------------------------------------------------------------*/

 FETCH_PARAM_PROCESSING: PROC ;

        CALL CTBFETCH( CSL_CMD_HANDLE,
                       CSL_RC,
                       CS_UNUSED,
                       CS_UNUSED,
                       CS_UNUSED,
                       FF_ROWS_READ );

        IF CSL_RC ^= CS_SUCCEED THEN
        DO ;
          MSGSTR       = 'CTBFETCH return parameter failed' ;
          NO_ERRORS_SW = FALSE ;
          CALL ERROR_OUT;
          CALL ALL_DONE ;
        END ;

        FF_ROW_NUM   = FF_ROW_NUM + 1;
        OR2_COL_RET1 = PM_PARAM1;

        RSLTNO(FF_ROW_NUM) = OUTPUT_ROW_STR2;

 END FETCH_PARAM_PROCESSING ;

/*--------------------------------------------------------------------*/
 /*                                                                    */
```

```
/* Subroutine to print output messages.                              */
/*                                                                   */
/*-------------------------------------------------------------------*/

ERROR_OUT: PROC;

        SAMP_RC   = CSL_RC;
        REST_TYPE = RF_TYPE ;

        IF DIAG_MSGS_INITIALIZED & ^BAD_INPUT
          THEN
            CALL GET_DIAG_MESSAGES ;

/*-------------------------------------------------------------------*/
/* display error message                                             */
/*-------------------------------------------------------------------*/

     MSG_TEXT_1 = TEST_CASE || SAMP_LIT  || SAMP_RC ||
                  REST_LIT  || REST_TYPE || '   '   ||
                  MSGSTR ;

        IF PRINT_ONCE THEN
          DO ;
            CALL DISP_DATA ;
            PRINT_ONCE = FALSE ;
          END ;

        NO_ERRORS_SW = FALSE ;
        MSGSTR       = BLANK ;
        SAMP_RC      = 0;
        REST_TYPE    = 0 ;

END ERROR_OUT;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to retrieve any diagnostic messages                    */
/*                                                                   */
/*-------------------------------------------------------------------*/
GET_DIAG_MESSAGES: PROC ;

DCL CNT        FIXED BIN(15) ;

/*-------------------------------------------------------------------*/
/* Disable calls to this subroutine                                  */
/*-------------------------------------------------------------------*/
```

```
        DIAG_MSGS_INITIALIZED = FALSE ;

/*-------------------------------------------------------------------*/
 /* First, get client messages                                       */
 /*-------------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_STATUS,
                      CS_CLIENTMSG_TYPE,
                      CS_UNUSED,
                      DG_NUM_OF_MSGS ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_STATUS CLIENTMSG_TYPE failed';
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;
          ELSE DO ;
            IF DG_NUM_OF_MSGS > 0 THEN
              DO ;
                DO CNT = 1 TO DG_NUM_OF_MSGS ;
                  CALL RETRIEVE_CLIENT_MSGS ;
                END ;
              END ;
          END ;

 /*-------------------------------------------------------------------*/
 /* Then, get server messages                                        */
 /*-------------------------------------------------------------------*/

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_STATUS,
                      CS_SERVERMSG_TYPE,
                      CS_UNUSED,
                      DG_NUM_OF_MSGS ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
          DO ;
            MSGSTR = 'CTBDIAG CS_STATUS SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
```

```
            CALL ALL_DONE ;
          END ;
          ELSE DO ;
            IF DG_NUM_OF_MSGS > 0 THEN
              DO ;
                DO CNT = 1 TO DG_NUM_OF_MSGS ;
                  CALL RETRIEVE_SERVER_MSGS ;
                END ;
              END ;
          END ;

END GET_DIAG_MESSAGES ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve diagnostic messages from client             */
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_CLIENT_MSGS: PROC ;

        I1 = 1 ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
                      CS_CLIENTMSG_TYPE,
                      DG_MSGNO,
                      CLIENT_MSG ) ;

        IF CSL_RC ^=  CS_SUCCEED THEN
        DO ;
          MSGSTR = 'CTBDIAG CS_GET CS_CLIENTMSG_TYPE FAILED' ;
          CALL ERROR_OUT ;
          CALL ALL_DONE ;
        END ;

/*--------------------------------------------------------------------*/
/* display message text                                               */
/*--------------------------------------------------------------------*/

        RSLTNO( I1 )      = 'Client Message:' ;
        I1                = 3 ;

        CM_SEVERITY_DATA = CM_SEVERITY ;
        CM_STATUS_DATA   = CM_STATUS ;
```

```
RSLTNO( I1 )      = CM_SEVERITY_HDR || CM_SEVERITY_DATA ||
                    CM_STATUS_HDR   || CM_STATUS_DATA ;
I1                = I1 + 1 ;

CM_OC_MSGNO_DATA = CM_MSGNO ;
RSLTNO( I1 )      = CM_OC_MSGNO_HDR || CM_OC_MSGNO_DATA ;
I1                = I1 + 1 ;

IF CM_MSGNO ^= 0 THEN
DO ;
    CM_OC_MSG_DATA     = SUBSTR( CM_TEXT, 1, 66 ) ;
    RSLTNO( I1 )       = '  OC MsgTx:  ' || CM_OC_MSG_DATA ;
    I1                 = I1 + 1 ;
    IF CM_TEXT_LEN > 66 THEN
    DO ;
        CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 67, 66 ) ;
        RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
        I1               = I1 + 1 ;
        IF CM_TEXT_LEN > 132 THEN
        DO ;
            CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 133, 66 ) ;
            RSLTNO( I1 )     = BLANK_13 ||
                               CM_OC_MSG_DATA_X ;
            I1               = I1 + 1 ;
            IF CM_TEXT_LEN > 198 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 199 ) ;
                RSLTNO( I1 )     = BLANK_13 ||
                                   CM_OC_MSG_DATA_X ;
                I1               = I1 + 1 ;
            END ;
        END ;
    END ;
END ;
ELSE DO ;
  RSLTNO( I1 ) = '  OC MsgTx:  No Message!' ;
  I1           = I1 + 1 ;
END ;

CM_OS_MSGNO_DATA = CM_OS_MSGNO ;
RSLTNO( I1 )      = '  OS MsgNo:  ' || CM_OS_MSGNO_DATA ;
I1                = I1 + 1 ;

IF CM_OS_MSGNO ^= 0 THEN
DO ;
    CM_OS_MSG_DATA     = SUBSTR( CM_OS_MSGTXT, 1, 66 ) ;
```

```
            RSLTNO( I1 )        = '  OS MsgTx:  ' ||
                                  CM_OS_MSG_DATA ;
            I1                  = I1 + 1 ;
            IF CM_OS_MSGTEXT_LEN > 66 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT, 67, 66 ) ;
                RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
                I1               = I1 + 1 ;
                IF CM_OS_MSGTEXT_LEN > 132 THEN
                DO ;
                    CM_OC_MSG_DATA_X = SUBSTR(CM_OS_MSGTXT,133,66);
                    RSLTNO( I1 )     = BLANK_13 || CM_OC_MSG_DATA_X ;
                    I1               = I1 + 1 ;
                    IF CM_OS_MSGTEXT_LEN > 198 THEN
                    DO ;
                        CM_OC_MSG_DATA_X = SUBSTR(CM_OS_MSGTXT,199);
                        RSLTNO( I1 )     = BLANK_13 ||
                                           CM_OC_MSG_DATA_X ;
                        I1               = I1 + 1 ;
                    END ;
                END ;
            END ;
        END ;
        ELSE DO ;
          RSLTNO( I1 ) = '  OS MsgTx:  No Message!' ;
          I1           = I1 + 1 ;
        END ;

END RETRIEVE_CLIENT_MSGS ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to retrieve diagnostic messages from server             */
/*                                                                    */
/*--------------------------------------------------------------------*/
RETRIEVE_SERVER_MSGS: PROC ;

        CALL CTBDIAG( CSL_CON_HANDLE,
                      CSL_RC,
                      CS_UNUSED,
                      CS_GET,
                      CS_SERVERMSG_TYPE,
                      DG_MSGNO,
                      SERVER_MSG ) ;

        IF CSL_RC ^= CS_SUCCEED THEN
```

```
          DO ;
            MSGSTR = 'CTBDIAG CS_GET CS_SERVERMSG_TYPE failed' ;
            CALL ERROR_OUT ;
            CALL ALL_DONE ;
          END ;

/*-------------------------------------------------------------------*/
/* display message text                                              */
/*-------------------------------------------------------------------*/

        SM_MSG_NO_DATA     = SM_MSGNO ;
        SM_SEVERITY_DATA   = SM_SEV ;
        SM_STATE_DATA      = SM_STATE ;
        SM_LINE_NO_DATA    = SM_LINE ;
        SM_STATUS_DATA     = SM_STATUS ;

        IF SM_SVRNAME_LEN > 66
          THEN
            SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 63 ) || '...' ;
          ELSE
            SM_SVRNAME_DATA = SUBSTR( SM_SVRNAME, 1, 66 ) ;

        IF SM_PROC_LEN > 66
          THEN
            SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 63 ) || '...' ;
          ELSE
            SM_PROC_ID_DATA = SUBSTR( SM_PROC, 1, 66 ) ;

        SM_MSG_DATA        = SUBSTR( SM_TEXT, 1, 66 ) ;
        RSLTNO (1)         = 'Server Message:' ;
        RSLTNO (3)         = SM_MSG_NO_HDR   || SM_MSG_NO_DATA ||
                             SM_SEVERITY_HDR || SM_SEVERITY_DATA ||
                             SM_STATE_HDR    || SM_STATE_DATA ;
        RSLTNO (4)         = SM_LINE_NO_HDR  || SM_LINE_NO_DATA ||
                             SM_STATUS_HDR   || SM_STATUS_DATA ;
        RSLTNO (5)         = SM_SVRNAME_HDR  || SM_SVRNAME_DATA ;
        RSLTNO (6)         = SM_PROC_ID_HDR  || SM_PROC_ID_DATA ;
        RSLTNO (7)         = SM_MSG_HDR      || SM_MSG_DATA ;

        IF SM_TEXT_LEN > 66 THEN
          DO ;
            SM_MSG_DATA_X = SUBSTR( SM_TEXT, 67, 66 ) ;
            RSLTNO(8)     = BLANK_13 || SM_MSG_DATA_X ;
            IF SM_TEXT_LEN > 132 THEN
              DO ;
                SM_MSG_DATA_X = SUBSTR( SM_TEXT, 133, 66 ) ;
```

```
                RSLTNO(9)      = BLANK_13 || SM_MSG_DATA_X ;
                IF SM_TEXT_LEN > 198 THEN
                  DO ;
                    SM_MSG_DATA_X = SUBSTR( SM_TEXT, 198 ) ;
                    RSLTNO(10)    = BLANK_13 || SM_MSG_DATA_X ;
                  END ;
              END ;
          END ;

END RETRIEVE_SERVER_MSGS ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to drop and to deallocate all handlers, to close        */
/* server connection and exit client library                          */
/*                                                                    */
/*--------------------------------------------------------------------*/
ALL_DONE: PROC ;

        CALL CLOSE_CONNECTION;
        CALL QUIT_CLIENT_LIBRARY;
        STOP ;

END ALL_DONE ;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to perform drop command handler, close server           */
/* connection, and deallocate Connection Handler.                     */
/*                                                                    */
/*--------------------------------------------------------------------*/
CLOSE_CONNECTION: PROC ;

/*--------------------------------------------------------------------*/
/* drop the command handle                                            */
/*--------------------------------------------------------------------*/

        CALL CTBCMDDR( CSL_CMD_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CTBCMDDROP CSL_CMD_HANDLE failed' ;
          CALL ERROR_OUT ;
        END ;
```

```
 /*--------------------------------------------------------------------*/
 /* close the server connection                                        */
 /*--------------------------------------------------------------------*/

         CALL CTBCLOSE( CSL_CON_HANDLE,
                        CSL_RC,
                        CS_UNUSED ) ;

         IF CSL_RC = CS_FAIL THEN
         DO ;
           MSGSTR = 'CTBCLOSE CSL_CON_HANDLE failed' ;
           CALL ERROR_OUT ;
         END ;

 /*--------------------------------------------------------------------*/
 /* DE_ALLOCATE THE CONNECTION HANDLE                                  */
 /*--------------------------------------------------------------------*/

         CALL CTBCONDR( CSL_CON_HANDLE,
                        CSL_RC ) ;

         IF CSL_RC = CS_FAIL THEN
         DO ;
           MSGSTR = 'CTBCONDROP CSL_CON_HANDLE failed' ;
           CALL ERROR_OUT ;
         END ;

 END CLOSE_CONNECTION ;

 /*--------------------------------------------------------------------*/
 /*                                                                    */
 /* Subroutine to perform exit client library and deallocate context */
 /* structure.                                                         */
 /*                                                                    */
 /*--------------------------------------------------------------------*/
 QUIT_CLIENT_LIBRARY: PROC ;

/*--------------------------------------------------------------------*/
 /* exit the Client Library                                           */
 /*--------------------------------------------------------------------*/

         CALL CTBEXIT( CSL_CTX_HANDLE,
                       CSL_RC,
                       CS_UNUSED ) ;

         IF CSL_RC = CS_FAIL THEN
```

```
        DO ;
          MSGSTR = 'CTBEXIT failed' ;
          CALL ERROR_OUT ;
        END ;

/*-------------------------------------------------------------------*/
/* de-allocate the context stricture                                 */
/*-------------------------------------------------------------------*/

        CALL CSBCTXDR( CSL_CTX_HANDLE,
                       CSL_RC ) ;

        IF CSL_RC = CS_FAIL THEN
        DO ;
          MSGSTR = 'CSBCTXDROP failed' ;
          CALL ERROR_OUT ;
        END ;

        EXEC CICS RETURN ;

END QUIT_CLIENT_LIBRARY ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Subroutine to display output                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/
DISP_DATA: PROC ;

        SDATEO  = TMP_DATE ;
        STIMEO  = TMP_TIME;
        PROGNMO = 'SYCTSAR4' ;

        SELECT( PAGE_CNT ) ;
          WHEN( 1 ) SPAGEO = '0001' ;
          WHEN( 2 ) SPAGEO = '0002' ;
          WHEN( 3 ) SPAGEO = '0003' ;
          WHEN( 4 ) SPAGEO = '0004' ;
          WHEN( 5 ) SPAGEO = '0005' ;
          WHEN( 6 ) SPAGEO = '0006' ;
          WHEN( 7 ) SPAGEO = '0007' ;
          WHEN( 8 ) SPAGEO = '0008' ;
          WHEN( 9 ) SPAGEO = '0009' ;
          OTHERWISE SPAGEO = '9999' ;
        END ;
```

```
        SERVERA = DFHBMPRO;
        SERVERO = PF_SERVER;

        USERA   = DFHBMPRO;
        USERO   = PF_USER;

        NETDRVA   = DFHBMPRO;
        NETDRVO   = PF_NETDRV;

        PSWDA   = DFHBMDAR;
        PSWDO   = PF_PWD;
        MSG1O   = MSG_TEXT_1;
        MSG2O   = MSG_TEXT_2;

/*-------------------------------------------------------------------*/
/* DISPLAY THE DATA                                                  */
/*-------------------------------------------------------------------*/

        EXEC CICS SEND MAP('A4PANEL')
                       MAPSET('SYCTBA4')
                       CURSOR
                       FRSET
                       ERASE
                       FREEKB ;

        EXEC CICS RECEIVE INTO(QF_ANSWER)
                          LENGTH(QF_LEN)
                          MAXLENGTH(QF_MAXLEN)
                          RESP(CICS_RESPONSE) ;

END DISP_DATA ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: NO_INPUT --- to handle MAPFAIL/ANYKEY condition            */
/*                                                                   */
/*-------------------------------------------------------------------*/
NO_INPUT:

        MSG_TEXT_1 = 'Please Enter Input Fields' ;
        GO TO GET_INPUT_AGAIN ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: GETOUT --- to handle CLEAR condition                       */
/*                                                                   */
```

```
/*-------------------------------------------------------------------*/
GETOUT:

        EXEC CICS RETURN ;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Label: ERRORS --- to handle ERROR condition                       */
/*                                                                   */
/*-------------------------------------------------------------------*/
ERRORS:

        EXEC CICS DUMP DUMPCODE('ERRS') ;

END SYCTSAR4;
```

APPENDIX C

Sybase Product Documentation by Audience

This appendix summarizes Mainframe Connect documentation by audience.

Note For instructions on ordering documentation, go to the Sybase web site at http://www.sybase.com.

Table C-1 lists the publications in the documentation set and shows the intended audience for each book. The symbols used in the table are:

- R = required for this role
- O = optional (can be useful for this role)

Table C-1: Documentation by audience

	Audience			
Title	**Mainframe systems support**	**Mainframe application developer**	**DirectConnect & Net-Gateway administrator**	**Application developer**
Mainframe Connect Client Option *Installation and Administration Guide*	R	R	R	R
Mainframe Connect Server Option for CICS *Installation and Administration Guide*	R		O	
Mainframe Connect Server Option for IMS and MVS *Installation and Administration Guide*	R		O	
Mainframe Connect Server Option *Programmer's Reference for COBOL*	R	R		
Mainframe Connect Server Option *Programmer's Reference for PL/1*	R	R		
Mainframe Connect Server Option *Programmer's Reference for RSPs*	R	R		
Mainframe Connect Client Option *Programmer's Reference for COBOL*	R	R		

Title	Audience			
	Mainframe systems support	**Mainframe application developer**	**DirectConnect & Net-Gateway administrator**	**Application developer**
Mainframe Connect Client Option *Programmer's Reference for PL/1*	R	R		
Mainframe Connect Client Option *Programmer's Reference for C*	R	R		
Mainframe Connect Client Option *Programmer's Reference for CSAs*	R	R		
Enterprise Connect Data Access and Mainframe Connect *Server Administration Guide* for DirectConnect	O		R	
Mainframe Connect DirectConnect for z/OS Option *Installation Guide*	O		R	
Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*	O		R	R
Mainframe Connect DirectConnect for z/OS Option *User's Guide for DB2 Access Services*	O		R	R

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V