SYBASE®

Programmer's Reference for PL/1

Mainframe Connect™ Server Option

15.0

IBM CICS, IMS, and MVS

DOCUMENT ID: DC36560-01-1500-01

LAST REVISED: July 2007

Copyright © 1991-2007 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

About This Book

This guide contains reference information for the PL/1 version of Open ServerConnect™ Gateway-Library.

Note The Open ServerConnect Gateway-Library is a subset of the generic Sybase® Gateway-Library.

Audience

This is a reference book for application programmers who write PL/1 programs that call Open ServerConnect Gateway-Library functions, as well as for system programmers who want to use its tracing and accounting features.

This book assumes that you are familiar with the PL/1 programming language and know how to write PL/1 programs under either CICS, MVS or IMS TM. It does not contain instructions for writing PL/1 programs. Rather, it describes the functions that can be called within your PL/1 programs to perform communication, conversion, tracing, and accounting functions.

How to use this book

This book contains these chapters:

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

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

- Chapter 2, "Topics," describes of 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, “Gateway-Library Quick Reference,” has a table of all Gateway-Library functions, their arguments and where they exist, and the symbolic constants used with each argument.
- Appendix B, “Sample RPC Application for CICS” contains a sample PL/1 application program that processes client RPC requests under CICS.
- Appendix C, “Sample Language Application for CICS,” has a sample PL/1 application program that processes client language requests under CICS
- Appendix D, “Sample RPC Application for IMS TM (Explicit),” provides a sample PL/1 application program that processes client RPC requests under the IMS TM implicit API.
- Appendix E, “Sample RPC Application for IMS TM (Implicit),” contains a sample PL/1 application program that processes client RPC requests under the IMS TM explicit API.
- Appendix F, “Sample Mixed-Mode Application,” has a sample PL/1 application program that includes both Gateway-Library and Client-Library function calls (a mixed-mode application).
- Appendix G, “Sample Tracing and Accounting Program,” provides a sample PL/1 program that demonstrates the use of all Gateway-Library tracing and accounting functions.

Product name changes

The following table describes new names for products in the 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 Client Connect 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

Old product name	New product name
DirectConnect™ for OS/390	DirectConnect for z/OS

The old product names are used throughout this book, except 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 Client Option for CICS *Installation and Administration Guide* – describes configuring the Enterprise Connect™ network, setting up APPC communications, installing the Server Option, setting up security, and troubleshooting for an IMS or z/OS environment.

- Mainframe Connect Server Option for CICS *Installation and Administration Guide* – describes configuring the Enterprise Connect network, installing the Server Option, setting up security, and troubleshooting for a CICS environment.

- Mainframe Connect Client Option for IMS and MVS *Installation and Administration Guide* – describes installing and configuring the Client Option, 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 DB2 UDB Option for CICS *Installation and Administration Guide* – describes configuring the mainframe, installing the DB2 UDB Option for CICS, setting up security, and troubleshooting for a CICS environment.

- Mainframe Connect DirectConnect for z/OS Option *Installation Guide* – describes installing a DirectConnect for z/OS Option server and service libraries.

- Enterprise Connect Data Access and Mainframe Connect *Server Administration Guide* for DirectConnect – describes administration of the DirectConnect for z/OS Option server. Information about administering specific service libraries and services is provided in other DirectConnect for z/OS Option publications.

- Mainframe Connect Client Option *Programmer's Reference for COBOL* – describes writing Client Option 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 Server Option 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 PL/1* – describes writing Client Option programs that call PL/1 Client-Library functions. This guide contains reference pages for Client-Library routines and descriptions of the underlying concepts for PL/1 programmers.
- Mainframe Connect Server Option *Programmer's Reference for PL/1* – provides reference material for writing Server Option programs that call PL/1 Gateway-Library functions. This guide contains reference pages for Gateway-Library routines and descriptions of the underlying concepts for PL/1 programmers.
- Mainframe Connect Client Option *Programmer's Reference for C* – describes writing Client Option 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 Client Option *Programmer's Reference for Client Services Applications* – provides information for anyone who designs, codes, and tests client services applications (CSAs).
- Mainframe Connect Server Option *Programmer's Reference for Remote Stored Procedures* – provides information for anyone who designs, codes, and tests remote stored procedures (RSPs).
- Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services* – describes configuring, controlling, and monitoring the DirectConnect for z/OS Option Transaction Router Service Library, as well as setting up security.
- Mainframe Connect DirectConnect for z/OS Option *User's Guide for DB2 Access Services* – describes configuring, controlling, and monitoring a DirectConnect for z/OS Option Access Service, as well as setting up security.
- Mainframe Connect Client Option and Server Option *Open ClientConnect and Open ServerConnect Messages and Codes* – provides details on messages that 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

Introduction

This chapter includes the following topics:

What is Open ServerConnect?

Open ServerConnect is a programming environment that lets you create mainframe transactions that Sybase client applications can execute. Open ServerConnect transactions can retrieve and update data stored on an IBM mainframe in any mainframe resource, such as VSAM files, TD queues, TS queues, and DL/1 databases, and in DB2 databases and other DBMSs.

Open ServerConnect is available for CICS, IMS TM and MVS. It runs on an IBM System/390 or plug-compatible mainframe computer. It uses a host transaction processor, such as CICS, as a communications front end and uses LU 6.2 or TCP/IP communications protocols.

For more information about Open ServerConnect, refer to the Mainframe Connect Server Option *Installation and Administration Guide*.

What is Gateway-Library?

Open ServerConnect provides a set of built-in, high-level functions for use in mainframe server applications that communicate with Sybase clients such as Open Client applications, third-party tools, and server-to-server programs. These built-in functions are linkable subroutines collectively known as Gateway-Library. Gateway-Library functions are called through a stub in the application program. An Open ServerConnect application program uses a CALL statement to invoke a Gateway-Library function.

You can use Gateway-Library functions with all versions of Open ServerConnect. Minor coding differences exist between CICS and IMS TM. Those differences are discussed in the section, "Differences between CICS, IMS, and MVS" on page 8.

Gateway-Library functions

Gateway-Library functions provide data conversion and LU 6.2 and TCP/IP communication functions to mainframe application programs. Each Gateway-Library function performs one or more specific task(s) in the communication between a server and a client.

Gateway-Library functions can:

- Retrieve and process requests from remote clients or servers
- Describe and return results to requesting clients or servers
- Manage global and transaction-specific tracing and accounting recording at the mainframe

Sybase Tabular Data Stream (TDS) protocol

Open ServerConnect uses the Sybase TDS protocol to transmit data between the mainframe server and Sybase clients. LU 6.2 or TCP/IP calls are embedded within Gateway-Library functions. All your application program needs to do to send and receive data streams is call the appropriate Gateway-Library functions. Because Gateway-Library functions automatically issue the appropriate LU 6.2 or TCP/IP calls, no additional code is needed. You do not need to know the details of TDS or your network protocol to use Gateway-Library functions.

Gateway-Library functions

All Gateway-Library functions begin with the letters "TD". For example, the TDINIT function initializes the Gateway-Library environment, and the TDRCVPRM function retrieves the data from a parameter in a call sent by a remote client.

The complete set of Gateway-Library functions is included on the product tapes. The program stubs that load and call the Gateway-Library functions are also included. For a list and description of all Gateway-Library functions, see Table 3-1 on page 65 in Chapter 3, "Functions."

Using Client/Server connections

Open ServerConnect supports both three-tier (gateway-enabled) and two-tier (gateway-less) environments. It can receive requests from LAN clients through any of the following:

- Transaction Router Service (TRS) or Net-Gateway™ using SNA or TCP/IP in a three-tier gateway-enabled environment
- TCP/IP in a two-tier gateway-less environment
- Adaptive Server Enterprise for server to server communication

Note If you use SNA as your protocol, use Online Transaction Processing (OLTP), or have large numbers of geographically-dispersed Adaptive Servers, you must use a TRS or Net-Gateway in a three-tier environment for routing.

For detailed information about compatibility, network drivers, new features in this version, performance factors, security, three-tier and two-tier environments, and how to install and configure Open ServerConnect in both environments, see the Mainframe Connect Server Option *Installation and Administration Guide*.

SNA connections

A group of logical connections is defined to SNA by the TRS administrator. Each logical connection connects a mainframe transaction processing region with a remote port on a TRS platform. Every request forwarded from a TRS to a mainframe server uses one of these logical connections to communicate with its remote partner. When a request is sent across a connection, it is called a *conversation*.

SNA connections are activated when a TRS is started and remain active until the TRS is shut down or deactivated.

TCP/IP connections

There is no difference in the use of Gateway-Library functions for SNA or TCP/IP networks.

In three-tier environments, the TRS administrator defines a group of TCP/IP communication sessions connecting a mainframe teleprocessing region with a remote port on a TRS. For detailed information about configuring TRS, see the Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*.

In two-tier environments, LAN clients directly login to Open ServerConnect using TCP/IP for connectivity. For more information, see the Mainframe Connect Server Option *Installation and Administration Guide*.

Initializing the Gateway-Library environment

Each mainframe server application that uses Gateway-Library must initialize the operating environment. Gateway-Library uses two structures to do this:

- IHANDLE
- TDPROC

IHANDLE

The *IHANDLE* structure is a transaction-wide structure that contains configuration parameters and other high-level information used to set up the operating environment for a Gateway-Library transaction. It is defined for each transaction by TDINIT.

Note TDINIT must be the first Gateway-Library function call in each application. The *IHANDLE* structure corresponds to the context handle in Open Client Client-Library™.

After the environment is initialized, an application must establish a conversation between the client and the server over one of the predefined connections. In Open ServerConnect, a logical connection is represented by a TDPROC structure. A TDPROC structure is associated with an *IHANDLE* structure and is defined in TDACCEPT.

TDPROC

The *TDPROC* structure corresponds to the *DBPROCESS* structure in DB-Library, as well as to the connection and command handles in Client-Library. Gateway-Library sends commands to the server and returns query results to the application through the *TDPROC* structure.

The handle for the *TDPROC* structure is stored in the argument *TDPROC*. Every Gateway-Library function that sends or accepts data across a connection must specify that connection handle in its *TDPROC* argument.

Starting and ending a conversation

A conversation is established when a client sends a transaction request and a server accepts the request. It remains open as long as the client and server are communicating about that request. When all results and messages are returned to the client, the program must end the conversation and free up the *TDPROC* structure. The function TDFREE is included for that purpose. The last Gateway-Library function called by your application must be TDTERM, which frees up any remaining storage.

After returning results to a client, a transaction can either end the communication (short transaction), or wait for another client request (long-running transaction). In long-running transactions, TDSNDDON marks the end of a single request, but does not necessarily end the transaction. To end a transaction, the *CONN_OPTIONS* argument of TDSNDDON must be set to *TDS_ENDRPC*. The transaction then calls TDFREE and TDTERM to free up storage. Long-running transactions can be coded under CICS or the IMS TM explicit API.

Handling client requests

Gateway-Library functions are designed to be symmetrical. That is, each time a program at one end of a connection issues a sending call, the program at the other end issues a corresponding receiving call.

In Open ServerConnect, the mainframe is always a server, never a client. Therefore, all the functions documented in this manual are those used by a server. Each TDRCV*xxx* function you code in your server application is responding to a corresponding send function issued by the client or TRS. Also, the data you send with a TDSND*xxx* function is accepted by a corresponding receive function in the client program.

For example, if the client is an Open ClientConnect program, TDRCVSQL and TDRCVPRM retrieve data sent by the client function CTBSEND, and TDSNDROW returns rows that are retrieved by the client function CTBFETCH.

Note It is possible to code mixed-mode programs that act as both server and client, using both Gateway-Library and Client-Library functions. To do this, you must have Open ClientConnect installed in the same region as Open ServerConnect.

Processing client requests

A client can send the following types of requests to a mainframe server:

- Remote Procedure Calls (RPCs)
- Language requests
- Cursor requests
- Dynamic SQL requests

Remote Procedure Calls (RPCs)

For each client RPC, the mainframe application programmer must write a corresponding server transaction that executes whenever the client calls that remote procedure.

Language requests

If you have MainframeConnect for DB2 UDB installed at the mainframe, you have a prewritten transaction that processes SQL language requests to DB2. This transaction, called AMD2, uses DB2 dynamic SQL to process incoming SQL statements. AMD2 handles all language request processing; no additional code is required.

If you do not have MainframeConnect for DB2 UDB, or if you want to send language requests to a custom-written language handler, you must write your own language transaction. Gateway-Library includes language-handling functions for this purpose. An example of a program that executes SQL language requests is included on the API tape (SYCPSAL1) and is printed in Appendix C, "Sample Language Application for CICS."

Note MainframeConnect is available only for MVS-CICS environments. For IMS TM and native MVS environments, use OmniSQL Access Module for DB2 for IMS TM and MVS with the SYRT transaction for processing language requests. Cursors and dynamic SQL are not supported.

Cursor requests

If MainframeConnect for DB2 UDB is installed at the mainframe, AMD2 processes cursor requests to DB2.

If MainframeConnect for DB2 UDB is not installed, you must write a server transaction to process cursor requests from the client. A single server transaction can process multiple cursor requests from the client.

Dynamic SQL requests

If MainframeConnect for DB2 UDB is installed at the mainframe, AMD2 processes dynamic requests to DB2.

If MainframeConnect for DB2 UDB is not installed, you must write a server transaction to process dynamic requests from the client. A single server transaction can process multiple dynamic requests from the client.

Differences between CICS, IMS, and MVS

For the most part, the use of Gateway-Library functions in CICS, IMS TM and MVS programs is the same. The minor differences that exist are discussed in Table 1-1 and noted in the reference pages for the affected functions.

Table 1-1: Coding differences between CICS, IMS TM and MVS

Function	Differences between CICS, IMS TM and MVS
TDINFRPC TDSTATUS	The action taken when the communication state (*COMM_STATE*) is TDS_RESET can differ between CICS, MVS and the IMS TM implicit API: • Under CICS, MVS, and the IMS TM explicit API, the transaction exits as soon as possible. • Under the IMS TM implicit API, the transaction can call TDGETREQ to accept another client request or it can exit.
TDINIT	The first argument differs between CICS and IMS TM: • Under CICS, the communications I/O block, passed as the first parameter in TDINIT, is the *EIB* (*DFHEIBLK*). • Under IMS TM, the first TDINIT parameter is *I/O PCB* (*IO_PCB*). • Under MVS, a null pointer should be used.
TDSETPT	Used with IMS TM only, to indicate the type of IMS TM transaction.
TDSNDDON	Value of *CONN_OPTIONS* in CICS, MVS, and the IMS TM explicit API can be set to TDS_ENDREPLY in long-running transactions. TDS_ENDREPLY cannot be used under the IMS TM implicit API. To learn how to simulate long-running transactions in the implicit API, see "Long-running transactions" on page 52.
TDINFACT TDSETACT	Accounting records are written to different logs under CICS, IMS TM and MVS: • Under CICS, accounting functions use VSAM files as log files. The default file name is *SYTACCT1*. • Under IMS TM, accounting functions use the IMS TM log. • Under MVS, the records are written to a sequential file. The *DDNAME* of this file is specified as a parameter in TDCUSTOM (MVSDDNAME).
TDINFLOG TDINFSPT TDLSTLSPT TDSETLOG TDSETSPT TDWRTLOG	Trace records are written to different logs under CICS, IMS TM and MVS: • Under CICS, tracing functions use VSAM files as log files. The default file name is *SYTDLOG1*. • Under IMS TM, tracing functions use the IMS TM log. • Under MVS, the records are written to a sequential file. The *DDNAME* of this file is specified as a parameter in TDCUSTOM (MVSDDNAME).

General processing procedures

Whether the incoming request is an RPC or a language, cursor, or dynamic request, the server application performs five general steps:

- Prepares the environment
- Accepts the request and retrieves the language, cursor, or dynamic request or RPC parameters
- Performs the requested action
- Returns results to the requesting client
- Ends the conversation

This section shows how to perform four of these tasks using Gateway-Library functions. The remaining task (the requested action) is performed using familiar programming procedures. See Chapter 3, "Functions" for detailed information about each function.

Note The tables in the following sections cover only the basic function sequences. Refer to the sample programs that come with Open ServerConnect and the examples to see how these functions are used in context.

Processing an RPC

When a client sends an RPC, a typical mainframe server application (short transaction) performs the tasks listed in Table 1-2. The arrows in the table indicate code loops.

Table 1-2: Functions to process RPCs

Task	Function
1. Prepare for incoming requests.	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
2. Accept the incoming request.	
Accept the incoming request. ←	TDACCEPT

Task	Function
3. Handle incoming parameters.	
Determine how many parameters were sent.	TDNUMPRM
Define variables for storing parameter information (datatype, length, data).	TDINFPRM
→ Retrieve the parameter (Loop until all parameters are retrieved).	TDRCVPRM
4. Process the request.	
Perform the requested task(s).	
5. Prepare to return results to the client.	
→ Set the length and address of each return parameter (Loop until all parameters are described).	TDSETPRM
→ Describe each column in a row to be returned (Loop until all columns are retrieved).	TDESCRIB
6. Return data to the client.	
→ Send data to the client, one row at a time (Loop until all rows are sent).	TDSNDROW
Send the return parameters, tell the client when results are finished, and close the connection.	TDSNDDON
7. End the conversation.	
Free the *TDPROC* structure.	TDFREE
Free the MVS storage (required with IMS TM; optional but recommended with CICS).	TDTERM

Processing a SQL language request

When a client sends a SQL select language request, a typical mainframe server application (short transaction) performs the tasks listed in Table 1-3. The arrows in the table indicate code loops.

Table 1-3: Functions to process language requests

Task	Function
1. Prepare for incoming requests.	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
2. Accept the incoming request.	
Accept the incoming request.	TDACCEPT

Task	Function
3. Handle incoming SQL statements.	
Determine the length (in bytes) of the incoming SQL string.	TDSQLLEN
→ Retrieve the SQL string (Loop until all parameters are retrieved).	TDRCVSQL
4. Process the request.	
Retrieve the requested data from the database.	
5. Prepare to return results to the client.	
→ Set the length and address of each return parameter (Loop until all parameters are described).	TDSETPRM
→ Describe each column in a row to be returned (Loop until all columns are retrieved).	TDESCRIB
6. Return data to the client.	
→ Send data to the client, one row at a time (Loop until all rows are sent).	TDSNDROW
Send the return parameters, tell the client when results are finished, and close the connection.	TDSNDDON
7. End the conversation.	
Free the *TDPROC* structure.	TDFREE
Free the MVS storage (required with IMS TM; optional but recommended with CICS).	TDTERM

Processing a cursor request

When a client sends a cursor request, a typical mainframe server application performs the tasks listed in Table 1-4.

Table 1-4: Functions to process cursor requests

Task	Function
1. Prepare for incoming requests.	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
2. Accept the incoming request.	
Accept the incoming request.	TDACCEPT
3. Determine the type of request.	
RPC, language, dynamic, or cursor.	TDINFPGM

Task	Function
4. Determine the type of cursor request.	
Retrieve *CURSOR_COMMAND*, *CURSOR_ID*, and *COMMAND_OPTIONS*.	TDCURPRO
5. Process the cursor request.	TDRCVSQL TDNUMPRM
Get SQL statement, number of parameters, table name, and parameters format. Either receive input parameters or update columns.	TDINFPRM TDRCVPRM
6. Describe the rows.	
Describe the rows. Send rows.	TDESCRIB TDSNDROW
7. Send return information.	
Send acknowledgment, *CURSOR_STATUS*, *CURSOR_INFO*.	TDCURPRO
8. Send DONE.	
Send a DONE package.	TDSNDDON
9. Accept the next request.	
Accept the incoming request.	TDGETREQ
10. End the conversation.	
Send final DONE package. Free the *TDPROC* structure. Free the MVS storage.	TDSNDDON TDFREE TDTERM

Processing a dynamic SQL request

When a client sends a dynamic request, a typical mainframe server application performs the tasks listed in Table 1-5.

Table 1-5: Functions to process dynamic requests

Task	Function
1. Prepare for incoming requests.	
Initialize the Gateway-Library environment. Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDINIT TDSETPT
2. Accept the incoming request.	
Accept the incoming request.	TDACCEPT

Task	Function
3. Determine the event type.	
Determine the type of client request: RPC, language, dynamic, or cursor.	TDINFRPC
4. Determine the type of dynamic operation.	
This can be a dynamic prepare request, dynamic execute request, dynamic execute immediate request, request for input or output parameter descriptions, or deallocate request.	TDYNAMIC
5. Process dynamic operation.	
Get the statement (for prepare). Get the statement ID (for all dynamic requests).	TDYNAMIC
6. Handle the incoming parameters.	
Determine number of parameters (for execute). Retrieve number of parameters (for execute). Retrieve input parameters (for execute).	TDNUMPRM TDINFPRM TDRCVPRM
7. Describe the data.	
Describe input/output parameters, or rows to be returned.	TDSNDROW
8. Return data to the client.	
Send result rows for execute or execute immediate. Send an acknowledge request. Send a done package.	TDYNAMIC TDSNDDON
9. Get the next request type.	
Accept the incoming request.	TDGETREQ
10. End the conversation.	
Send final DONE package. Free the *TDPROC* structure. Free the MVS storage.	TDSNDDON TDFREE TDTERM

Processing a long-running transaction

When a client sends a series of RPCs, a typical mainframe server application (long-running transaction) performs the tasks listed in Table 1-6. The arrows in the table indicate code loops.

Table 1-6: Functions to process long-running transactions

Task	Function
1. Prepare for incoming requests.	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
2. Accept the incoming request.	
Accept the incoming request.	TDACCEPT
3. Handle incoming parameters.	
Determine how many parameters were sent.	TDNUMPRM
→ Loop until all parameters are retrieved.	TDINFPRM TDRCVPRM
4. Process the request.	
Perform the requested task(s).	
5. Prepare to return results to the client.	
→ Loop to describe return parameters and columns in return rows.	TDSETPRM TDESCRIB
6. Return data to the client	
→ Send rows and return parameters to the client (For final TDSNDDON, you must Set *STATUS* to TDS-DONE-FINAL and *CONN_OPTIONS* to TDS-ENDREPLY).	TDSNDROW TDSNDDON
7. Accept next request.	
Accept the incoming request.	TDGETREQ
8. Repeat steps 3-6.	
Repeat steps 3–6 for each successive request (For final TDSNDDON, you must set *STATUS* to TDS-DONE-FINAL and *CONN_OPTIONS* to TDS-ENDRPC).	
9. End the conversation.	
Free the *TDPROC* structure.	TDFREE
Free the MVS storage (required with IMS TM; optional but recommended with CICS).	TDTERM

Additional processing options

Table 1-7 on page 15 contains additional Gateway-Library functions for occasional use.

Table 1-7: Functions for process options

Task	Function
Locate the decimal point for each column.	TDSETBCD
Convert mainframe datatypes to those used by DB-Library programs.	TDCONVRT
Return information about the currently running transaction.	TDINFPGM
Return information about the current client request.	TDINFRPC
Return information about the user-defined datatype associated with a column.	TDINFUDT
Change data length of a column before sending the row.	TDSETLEN
Specify the Gateway-Library datatype for a column.	TDSETUDT
Send error or informational messages to the client.	TDSNDMSG
Retrieve status information about the connection.	TDSTATUS
Retrieve client login information.	TDGETUSR

Tracing and accounting functions

Table 1-8 contains Gateway-Library functions that are available for tracing and accounting. These functions are used primarily by a system programmer.

Table 1-8: Functions for tracing and accounting

Task	Function
Turn mainframe-based tracing on or off.	TDSETLOG
Write a user-defined or system entry to the trace/error log.	TDWRTLOG
Return trace setting information.	TDINFLOG
Set tracing on or off for a specified transaction.	TDSETSPT
List all transactions for which specific tracing is enabled.	TDLSTSPT
Indicate whether tracing is on or off for a specified transaction.	TDINFSPT
Turn mainframe-based accounting on or off.	TDSETACT
Determine whether accounting is on and the name of the accounting log file.	TDINFACT

CHAPTER 2

Topics

This chapter presents information about Open ServerConnect concepts and procedures that are grouped by topics listed in alphabetical order.

This chapter contains the following topics:

Character sets

Open ServerConnect can accept requests in a variety of client character sets. The client identifies the character set in the login packet, which is forwarded to the mainframe server. Open ServerConnect does the necessary translations (for example, from ASCII 8 to EBCDIC or from Shift-JIS to IBM Kanji).

Gateway-Library uses translate tables and conversion modules to convert workstation characters into characters used by the mainframe. A Japanese Conversion Module (JCM) is available with Open ServerConnect on a separate tape. A system programmer can customize translate tables or rename the JCM at your site. See the Mainframe Connect Server Option *Installation and Administration Guide* for details.

- Supported workstation character sets
- Supported mainframe character sets

Supported workstation character sets

The tables in this section list the supported character sets, indicate whether or not each set can be used for Japanese characters, and indicate whether those that can be used for Japanese characters allow hankaku katakana (single-byte characters).

Single-byte character sets

Table 2-1 lists the single-byte character sets (SBCSs) that are supported at the workstation.

Table 2-1: SBCSs supported at the workstation

Character set name	Supports Japanese characters?	Includes Hankaku Katakana?
iso_1	No	Not applicable
cp850	No	Not applicable
cp437	No	Not applicable
roman8	No	Not applicable
mac	No	Not applicable
ascii_8	No	Not applicable
sjis	Yes	Yes

As shipped, the default character set on most workstations is iso_1; on an IBM RS/6000, it is cp850; and on HP platforms, it is roman8. Refer to your workstation documentation to find out what character set is supported on your workstation.

Double-byte character sets

Table 2-2 lists the double-byte character sets (DBCS) that are supported at the workstation.

Table 2-2: DBCS supported at the workstation

Character set name	Includes Hankaku Katakana?
eucjis	Yes
deckanji	No

Note All supported DBCS can be used for Japanese characters.

Supported mainframe character sets

Table 2-3 lists the character sets that are supported at the mainframe.

Table 2-3: Character sets supported at the mainframe

Type of character set	Character set name	Includes Hankaku Katakana?
Single-byte	EBCDIC	Yes, in Japan only
Double-byte	IBM Kanji	No

Note Although single-byte characters can be read as either hankaku katakana or lowercase alphabetic characters, only one option can be specified in a single CICS region.

Communication states

Clients and servers that use LU 6.2 and TCP/IP to communicate with each other are said to be in one of three communication states:

- SEND – Program can send information to the client
- RECEIVE – Program can receive information from the client
- RESET – Connection is closed

At any given time, communication between the mainframe server and the client is in only one direction. The mainframe server can send information to a client or receive information from a client, but not both.

The mainframe server must be in the correct communication state before it can execute certain functions. For example, it must be in SEND state to send results or messages; it must be in RECEIVE state to retrieve requests.

The communication state of the mainframe server is set by the Gateway-Library functions. For example, after it accepts a client request with TDACCEPT, an application switches to SEND state, because the next communication it has with the client is to send results. In most cases, the required communication state is evident from the name of the function. The reference pages in Chapter 3, "Functions" explain which state is required for each function.

After a client request is processed and all results returned, an application calls TDSNDDON. If the transaction processes only a single client request (a short transaction), TDSNDDON ends communication with the client. If the transaction is a long-running transaction that finished one client request and is awaiting another, TDSNDDON keeps communication open and switches the communication state from SEND to RECEIVE. See "Long-running transactions" on page 52 for details about this type of transaction.

When you follow the usual sequence of function calls, the mainframe server is always in the correct state. To check the communication state before issuing a function call, call TDSTATUS. If your application tries to execute a function when the mainframe server program is not in the appropriate communication state, the operation fails, and the return code indicates that the application is in the wrong state.

Cursors

Open ServerConnect supports cursor transactions. The Sybase generic Open Client libraries (DB-Library and Client-Library), Adaptive Server, and Open Server support cursors. Cursor support at the mainframe allows clients using these products to include cursors when accessing mainframe data.

Note Open ClientConnect versions 2.0 and 3.x do not support cursors.

This section contains the following subsections:

What is a cursor?

A cursor is a symbolic name that is linked with a SQL statement. Declaring a cursor establishes this link. The SQL statement can be one of the following:

- A SQL select statement
- A Transact-SQL execute statement
- A dynamic SQL prepare statement

The SQL statement associated with a cursor is called the body of the cursor. When a client opens a cursor, it executes the body of the cursor, which in turn generates a result set. The Open ServerConnect application is responsible for detecting cursor requests and passing cursor results back to the client.

Benefits of using cursors

Cursors allow a client application to retrieve and change data in a powerful, flexible manner. They allow applications to access and move around in a set of data rows, rather than merely retrieve a complete result set.

Moreover, a single connection can have multiple cursors open at the same time. All of the cursor result sets are simultaneously available to the application, which can fetch them at will. This is in direct contrast to other types of result sets, which must be handled one row at a time, in a sequential fashion.

Further, a client application can update underlying database tables while actively fetching rows in a cursor result set.

How cursors work in Open ServerConnect

The following steps show how Open ServerConnect handles a cursor request:

1 Open ServerConnect receives a client request.

2 The Gateway-Library transaction determines the type of request by calling TDINFPGM and checking the value of the *REQUEST_TYPE* argument.

3 If the type of request is TDS_CURSOR_EVENT, the transaction calls TDCURPRO to determine what cursor command the client sent and which cursor is affected. The transaction then processes the command and returns results to the client.

Types of cursor commands

Table 2-4 summarizes the types of cursor commands a client can issue. The "Command" column in the table shows the value in the *CURSOR_COMMAND* field of the *CURSOR_DESC* structure.

Table 2-4: Summary of cursor commands

Command	Action
TDS_CURSOR_DECLARE 0x001	Associate a cursor ID with the body of the cursor.
TDS_CURSOR_OPENCMD 0x002	Execute the body of the cursor and generate a cursor result set.
TDS_CURSOR_FETCH 0x003	Fetch rows from the cursor result set.
TDS_CURSOR_DELETE 0x004	Delete the contents of the current cursor row.
TDS_CURSOR_UPDATE 0x005	Update the contents of the current cursor row.
TDS_CURSOR_INFO 0x006	Report the status of the cursor, or set the cursor row fetch count.
TDS_CURSOR_CLOSE 0x007	Make the cursor result set unavailable. Reopening a cursor regenerates the cursor result set.
TDS_CURSOR_DEALLOC 0x008	Render the cursor nonexistent. A cursor that is deallocated cannot be re-opened.

A typical client application issues cursor commands in the order in which they are listed in Table 2-4 on page 22, but the order can vary. For example, a client can fetch against a cursor, close the cursor, then reopen and fetch it again.

Because a client and server can exchange information about multiple cursors in a single connection session, they need a means of uniquely identifying each cursor. An Open ServerConnect application responds to a cursor declaration by sending back a unique cursor ID. The ID is an integer. The client and the server refer to the cursor by this ID for the lifetime of the cursor.

Declare cursor

When the cursor command is TDS_CURSOR_DECLARE, the client is declaring a new cursor. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine: • The type of cursor command (DECLARE, in this case), and • Whether or not this cursor can be used to update database tables.
TDNUMPRM	Retrieve the number of parameters associated with the cursor.
TDINFPRM	Get the format of each associated parameter (once for each parameter).
TDRCVSQL	Retrieve the SQL text associated with the cursor.
[application logic]	[Declare the cursor to the application]
TDCURPRO	Assign a cursor ID to the cursor.
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Open cursor

When the cursor command is TDS_CURSOR_OPENCMD, the client is executing the body of the cursor and generating a cursor result set. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine the type of cursor command (OPEN, in this case).
TDNUMPRM	Retrieve the number of parameters associated with the cursor.
TDRCVPRM	Retrieve cursor parameters (once for each parameter).
TDCURPRO	Send the cursor status to the client.
TDESCRIB	Describe column results to the client (once for each column).
TDSETUDT	[optional] Set user datatype, if needed.
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Fetch rows

When the cursor command is TDS_CURSOR_FETCH, the client is fetching a row through a cursor. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine the type of cursor command (FETCH).
[application logic]	[Adjust the cursor]
TDSNDROW	Send back *n* rows of results.
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Delete cursor

When the cursor command is TDS_CURSOR_DELETE, the client is deleting the current cursor row. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine the type of cursor command.
[application logic]	[Adjust the cursor]
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Update cursor

When the cursor command is TDS_CURSOR_UPDATE, the client is updating the contents of the current cursor row. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine the type of cursor command.
TDRCVSQL	Retrieve the SQL text associated with the cursor.
[application logic]	[Adjust the cursor]
TDSNDDON	Send the reply to the client.

Request cursor status

The Client-Library command ct_cmd_props can request information on cursor options, identifiers, and status. When the cursor command is TDS_CURSOR_INFO with the option CUR_ASKSTATUS (ct_cmd_props), the client is requesting the status of the cursor. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine the type of cursor command.
TDCURPRO	Send the cursor status to the client.
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Get fetch count

The Client-Library command ct_cursor can request cursor row information. When the cursor command is TDS_CURSOR_INFO with the option CUR_SETROW (ct_cursor (CS_CURSOR_ROW)), the client is setting the row fetch count. In response, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine: • The type of cursor command, and • The number of rows to be returned with each fetch command.
[application logic]	[Adjust the cursor]
TDCURPRO	Send the cursor status to the client.
TDGETREQ	Retrieve the next part of the cursor request.

Close cursor or deallocate cursor

When the cursor command is TDS_CURSOR_CLOSE, the client is requesting to close a cursor. This can be a request to both close and deallocate the cursor, or to close it only.

When the cursor command is TDS_CURSOR_DEALLOC, the client is requesting to deallocate a cursor.

In response to either command, the Gateway-Library transaction calls the following functions:

Function	Action
TDCURPRO	Determine: • The type of cursor command, and • Whether the cursor should also be deallocated.
[application logic]	[Close the cursor]
TDCURPRO	Send the cursor status to the client.
TDSNDDON	Send the reply to the client.
TDGETREQ	Retrieve the next part of the cursor request.

CURSOR_DESC structure

A *CURSOR_DESC* structure contains information about a cursor, including the following:

- The cursor's unique ID
- The type of cursor command most recently issued by the client
- The status of the cursor

A *CURSOR_DESC* structure is defined in SYGWPLI as follows:

```
CURSOR_ID              FIXED BIN(31),
NUMBER_OF_UPDATE_COLUMNS FIXED BIN(31),
FETCH_COUNT            FIXED BIN(31),
CURSOR_STATUS          FIXED BIN(31),
CURSOR_COMMAND         FIXED BIN(31),
COMMAND_OPTIONS        FIXED BIN(31),
FETCH_TYPE             FIXED BIN(31),
ROW_OFFSET             FIXED BIN(31),
CURSOR_NAME_LENGTH     FIXED BIN(31),
CURSOR_NAME            CHAR(30),
TABLE_NAME_LENGTH      FIXED BIN(31),
TABLE_NAME             CHAR(30);
```

Fields in a CURSOR_DESC structure

Table 2-5 describes each field in a *CURSOR_DESC* structure.

Table 2-5: Fields in a CURSOR_DESC structure

Field name	Description	Notes
CURSOR_ID	The current cursor identifier.	The Open ServerConnect application must set CURSOR_ID when responding to a TDS_CURSOR_DECLARE (DECLARE CURSOR) command from the client. This happens when the client sends a DECLARE CURSOR command that has *CURSOR-NAME* as a required parameter. The Gateway-Library transaction receives the DECLARE CURSOR command from the client, calls TDCURPRO to specify a unique cursor identifier in the CURSOR_ID field, and returns the unique cursor ID to the client. The client uses the unique cursor ID (instead of the initial cursor name) in the CURSOR_ID field of the *CURSOR_DESC* structure for all subsequent commands regarding this cursor.
NUMBER_OF_ UPDATE_ COLUMNS	The number of columns in a cursor update clause.	NUMBER_OF_UPDATE_COLUMNS is set to 0 if there are no update columns. This information is available at declare time.
FETCH_COUNT	The current row fetch count for this cursor (the number of rows that are sent to the client in response to a TDS_CURSOR_ FETCH command).	FETCH_COUNT is described when a TDS_CURSOR_INFO command is received from the client, or sent to the client in response to such a command. FETCH_COUNT is set to 1 if the client has not explicitly set a row fetch count. If the Open ServerConnect application cannot support the requested fetch count, it can set this field to a different value before responding.
CURSOR_STATUS	The status of the current cursor.	Open ServerConnect sets the cursor status in response to the cursor command received from the client. See Table 2-6 on page 28 for a list of legal values.
CURSOR_ COMMAND	The current cursor command type.	See Table 2-7 on page 29 for a list of legal values.
COMMAND_ OPTIONS	Any options associated with the cursor command.	Not all commands have associated options. The value of COMMAND_OPTIONS depends on the cursor command. Table 2-7 on page 29 describes the possible values for COMMAND_OPTIONS.

Field name	Description	Notes
FETCH_TYPE	The type of fetch requested by a client.	FETCH_TYPE is described when a TDS_CURSOR_FETCH command is received from the client. The valid fetch types and their meanings are as follows: - TDS_NEXT – next row - TDS_PREV – previous row - TDS_FIRST – first row - TDS_LAST – last row - TDS_ABSOLUTE – row identified in the ROW_OFFSET field - TDS_RELATIVE – current row plus or minus the value in the ROW_OFFSET field Requests to Open ServerConnect always have a FETCH_TYPE of TDS_NEXT.
ROW_OFFSET	The row position for TDS_ABSOLUTE or TDS_RELATIVE fetches.	ROW_OFFSET is undefined for all other fetch types. ROW_OFFSET is described when a TDS_CURSOR_FETCH command is received from the client.
CURSOR_NAME_ LENGTH	The length of the cursor name in CURSOR_NAME.	CURSOR_NAME_LENGTH is zero if not used. If used, CURSOR_NAME_LENGTH is the actual length.
CURSOR_NAME	The name of the current cursor.	
TABLE_NAME_ LENGTH	The length of the table name in TABLE_NAME.	TABLE_NAME_LENGTH is zero if not used. If used, TABLE_NAME_LENGTH is the actual length. TABLE_NAME_LENGTH is described when a TDS_CURSOR_UPDATE or TDS_CURSOR_DELETE command is received from the client.
TABLE_NAME	The table name associated with a cursor update or delete command.	TABLE_NAME is described when a TDS_CURSOR_UPDATE or TDS_CURSOR_DELETE command is received from the client.

Values for *CURSOR_STATUS*

The CURSOR_STATUS field of the *CURSOR_DESC* structure is a bit mask that can take any combination of the values described in Table 2-6.

Table 2-6: Values for CURSOR_STATUS (CURSOR_DESC)

CURSOR_COMMAND value	Meaning
TDS_CURSTAT_DECLARED	The cursor is declared. This status is reset after the next cursor command is processed.
TDS_CURSTAT_OPEN	The cursor is open.

CURSOR_COMMAND value	Meaning
TDS_CURSTAT_ROWCNT	The cursor specified the number of rows that should be returned for the TDS_CURSOR_FETCH command.
TDS_CURSTAT_RDONLY	The cursor is read only; it cannot be updated. The Open ServerConnect application should return an error to the client if TDS_CURSOR_UPDATE or TDS_CURSOR_DELETE is received for this cursor.
TDS_CURSTAT_UPDATABLE	The cursor can be updated.
TDS_CURSTAT_CLOSED	The cursor is closed, but not deallocated. It can be opened again later. This status is also set upon declaration of a cursor. Open ServerConnect clears it when a TDS_CURSOR_OPEN is received and resets it when a TDS_CURSOR_CLOSE is received.
TDS_CURSTAT_DEALLOC	The cursor is closed and deallocated. No other status flags should be set at this time.

Values for *CURSOR_COMMAND* and *COMMAND_OPTIONS*

The CURSOR_COMMAND field of the *CURSOR_DESC* structure indicates the command to be processed. It can take one of the values described in Table 2-7 on page 29. TDCURPRO can update this field with the next command to process for a given cursor. Table 2-7 also lists the relevant COMMAND_OPTIONS values.

Table 2-7: Values for CURSOR_COMMAND (CURSOR_DESC)

CURSOR_COMMAND value	Meaning	Legal values for COMMAND_OPTIONS
TDS_CURSOR_CLOSE	Cursor close command.	TDS_CURSOR_DEALLOC or TDS_CURSOR_UNUSED. TDS_CURSOR_DEALLOC indicates that the cursor will never be reopened. The Open ServerConnect application should delete all associated cursor resources. The cursor ID number can be reused.
TDS_CURSOR_DECLARE	Cursor declare command. The application can obtain the actual text of the cursor statement through TDRCVSQL.	TDS_CURSOR_UPDATABLE, TDS_CURSOR_RDONLY, or TDS_CURSOR_DYNAMIC. TDS_CURSOR_DYNAMIC indicates that the client is declaring the cursor against a dynamically prepared SQL statement. In this case, the text of the cursor statement is actually the name of the prepared statement.

CURSOR_COMMAND value	Meaning	Legal values for COMMAND_OPTIONS
TDS_CURSOR_DELETE	Cursor delete command. Performs a positional row delete through a cursor.	There are no valid options for this command. COMMAND_OPTIONS always has the value TDS_CURSOR_UNUSED.
TDS_CURSOR_FETCH	Cursor fetch command. Performs a row fetch through a cursor.	There are no valid options for this command. COMMAND_OPTIONS always has the value TDS_CURSOR_UNUSED.
TDS_CURSOR_INFO	Cursor information command. The client sends this command to the Open ServerConnect application to set the cursor row fetch count or to request cursor status information. The Open ServerConnect application sends this command to the client in response to any cursor command (including TDS_CURSOR_INFO itself) to describe the current cursor.	TDS_CURSOR_SETROWS when the client is describing the current row fetch count. The FETCH_COUNT field contains the requested fetch count. TDS_CURSOR_ASKSTATUS when the client is requesting status information about the current cursor. This generally occurs when the client sends an attention and wants to see which cursors are still available afterwards. The CURSOR_ID field contains 0. The Open ServerConnect application should send back a TDS_CURSOR_INFO response for each cursor currently available. TDS_CURSOR_INFORMSTATUS when the Open ServerConnect application is responding to a TDS_CURSOR_INFO command. The CURSOR_STATUS field contains the cursor status.
TDS_CURSOR_OPEN	Cursor open command.	TDS_CURSOR_HASARGS or TDS_CURSOR_UNUSED.
TDS_CURSOR_UPDATE	Cursor update command. Performs a positional row update through a cursor. The Open ServerConnect application can obtain the actual text of the cursor update statement by calling TDRCVSQL.	TDS_CURSOR_UNUSED.

Handling cursor requests

An Open ServerConnect application uses a TDS_CURSOR_EVENT to handle cursor requests. The handler includes code to detect which of the cursor commands was issued and to respond with the appropriate information.

The first task inside the event handler is to determine the current cursor and the cursor command that triggered the TDS_CURSOR_EVENT. It does this by calling TDCURPRO with the *ACTION* argument set to TDS_GET. Open ServerConnect fills the *CURSOR_COMMAND* field of the Open ServerConnect application *CURSOR_DESC* structure with the command type.

The application can then decide what other information it needs to retrieve, if any, as well as what data to send back to the client. In some cases, it may need to retrieve parameter formats and parameters; in others, it may want to know the status of the current cursor and the number of rows to fetch. It may only need to send back a TDS_CURSOR_INFO command, or it may need to send back result data or return parameters.

How to respond to specific cursor requests

This section contains information on how a TDS_CURSOR_EVENT handler should respond to specific types of cursor requests.

On each cursor declare request, the Open ServerConnect application must set a unique cursor identifier before TDCURPRO, with *ACTION* set to TDS_SET. Open ServerConnect sets CURSOR_STATUS and CURSOR_COMMAND in the *CURSOR_DESC* structure.

Table 2-8 summarizes the valid exchange of cursor requests and responses between a client and an Open ServerConnect application. The forward arrow indicates that *ACTION* is set to TDS_GET, which means the application is retrieving information from the client. The backward arrow indicates that *ACTION* is set to TDS_SET, which means the application is sending information to the client.

Table 2-8: Valid cursor requests and responses

Client action	Open ServerConnect application response
Declares a cursor. (CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_DECLARE.	→ Retrieve CURSOR_COMMAND value from *CURSOR_DESC*. (TDCURPRO) → Retrieve number of cursor parameters, if any. (TDNUMPRM) → Retrieve format of cursor parameters, if any. (TDINFPRM) → Retrieve actual text of cursor command. (TDRCVSQL) ← Set cursor ID. Set CURSOR_ID field to unique cursor ID. (TDCURPRO) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.
Requests the status of the current cursor or sends a fetch count. (CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_INFO.	→ Retrieve CURSOR_COMMAND, CURSOR_ID, and COMMAND_OPTIONS values from *CURSOR_DESC* structure. (TDCURPRO) ← Send number of rows to be returned per fetch, if client set COMMAND_OPTIONS field to TDS_CURSOR_SETROWS. (TDCURPRO) ← Send status of all available cursors, if client set COMMAND_OPTIONS field to TDS_CURSOR_ASKSTATUS. Set CURSOR_ID field to cursor ID. (TDCURPRO once for each active declared, opened or closed cursor.) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.) *Note:* If the client request is ct_cmd_props with cursor options, then the CURSOR_COMMAND field is TDS_CURSOR_INFO with TDS_CURSOR_ASKSTATUS option. If the client request is ct_cursor (CS_CURSOR_ROWS), then the *CURSOR_COMMAND* field is TDS_CURSOR_INFOR with TDS_CURSOR_SETROWS option.

Client action	Open ServerConnect application response
Opens a cursor. (CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_OPEN.	→ Retrieve CURSOR_COMMAND and CURSOR_ID values from *CURSOR_DESC* structure. (TDCURPRO) → Retrieve number of cursor parameters, if any. (TDNUMPRM) → Retrieve format of cursor parameters and actual parameters, if any. (TDINFPRM, TDRCVPRM) ← Send cursor status. Set CURSOR_ID to current cursor ID. (TDCURPRO) ← Describe result row formats. (TDESCRIB with *TYPE* argument set to TDS_ROWDATA) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.
Fetches rows. (CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_FETCH.	→ Retrieve CURSOR_COMMAND and CURSOR_ID values from *CURSOR_DESC* structure. (TDCURPRO) ← Send result rows, FETCH_COUNT times. (TDSNDROW) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.
Sends a cursor close command. (CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_CLOSE.	→ Retrieve CURSOR_COMMAND and CURSOR_ID values from *CURSOR_DESC* structure. (TDCURPRO) ← Send cursor status. Open ServerConnect sets cursor status, not the application. (TDCURPRO) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.

Client action	Open ServerConnect application response
Updates a cursor. CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_UPDATE.	→ Retrieve CURSOR_COMMAND and CURSOR_ID values from *CURSOR_DESC* structure. (TDCURPRO) → Retrieve actual text of cursor command. (TDRCVSQL) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL.
Deletes a cursor. CURSOR_COMMAND field of *CURSOR_DESC* contains TDS_CURSOR_DELETE.	→ Retrieve CURSOR_COMMAND and CURSOR_ID values from *CURSOR_DESC* structure. (TDCURPRO) ← Send a DONE packet. (TDSNDDON with *STATUS* argument set to TDS_DONE_FINAL)

Additional information:

- The Open ServerConnect application response to a cursor command always concludes with a call to TDSNDDON with a *STATUS* argument of TDS_DONE_FINAL.
- After the Open ServerConnect application issues the first TDCURPRO command with *ACTION* set to TDS_SET, any further information the application sends applies to this cursor until a TDSNDDON with a *STATUS* argument of TDS_DONE_FINAL is issued.
- Internally, Open ServerConnect replaces the parameter formats received when the client declares a cursor with those received when the client opens a cursor. This is necessary if the format of the parameter passed is not exactly the same as that of the parameter declaration. For example, a parameter may be declared as a TDS_INIT, but the parameter being passed when the cursor is opened may be of type TDS_SMALLINT.
- In response to a TDS_CURSOR_FETCH command, TDSNDROW sends a single row of data, and should be called as many times as the number in the current cursor's row fetch count.

Processing cursor requests

The Open ServerConnect application program uses TDINFPGM and TDGETREQ to determine what type of request the client sent. For cursor requests, the application processes cursor commands and generates result sets.

Multiple cursor commands per transaction invocation are not allowed because TRS can only pass one cursor command per TDS_CURSOR event. To process multiple commands, use the Open ServerConnect long-running transaction and accept each new command request with TDGETREQ.

Cursors are limited to SQL statements and cannot be used with other types of languages.

Cursor support is not available for Japanese or DBCS.

For example, if a client sends an OPEN CURSOR request to a DB2 application, the Open ServerConnect application is responsible for defining and executing the actual DB2 OPEN CURSOR command. Open ServerConnect is merely the transport mechanism for cursor commands.

Customization

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

Gateway-Library functions use the following customized items:

- 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 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 U.S. English.
- Support for DBCSs

  The customization module indicates whether DBCSs are supported:

  - If DBCSs are supported, this module indicates whether single-byte characters are treated as lower-case alphabetic characters or as single-byte (hankaku) katakana during DBCS processing
  - If DBCSs are not used, this module specifies the name of the default single-byte character set to be used at the mainframe
- DB2 LONG VARCHAR data strings with lengths greater than 255
- Dynamic network drivers

The customization module sets up support for the following network drivers:

- LU 6.2
- IBM TCP/IP
- CPIC
- Interlink TCP/IP

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

To retrieve customization information, call TDGETUSR.

Datatypes

Open ServerConnect supports a wide range of datatypes. These datatypes, named TDS*xxx*, are compatible with DB2 datatypes, Client-Library datatypes, and DB-Library datatypes.

When either the Open ServerConnect or the Japanese Conversion Module (JCM) receives a client request, it automatically converts some DB-Library and Client-Library datatypes to Open ServerConnect datatypes. When either returns results to the client, it converts them back. See the conversion tables in the "Comments" sections of the reference pages in Chapter 3, "Functions" to find the datatype conversions performed by specific functions.

For most datatypes, Open ServerConnect or the JCM does workstation-to-mainframe and character set translations when retrieving incoming requests, then translates the datatypes back to workstation datatypes before returning results. For binary datatypes, both pass the data through. Table 2-9 on page 37 indicates which datatypes are passed through without translation.

Datatype descriptions and correspondences

Open ServerConnect supports a subset of Client-Library and DB-Library datatypes. Table 2-9 on page 37 lists those datatypes and their Gateway-Library equivalents.

Table 2-9: Open ServerConnect datatypes

Open ServerConnect datatype and PL/1 data description	Client-Library/C and DB-Library datatypes	Datatype description
TDSBINARY 01 BINVAL CHAR(1)	CS_BINARY DBBINARY	Fixed binary type. No translations are performed on this datatype.
TDSBIT 01 BITVAL BIT (01)	CS_BIT DBBIT	Single bit type. Used for Boolean values. Used with PL/1 only; not supported for COBOL. No translations are performed on this datatype.
TDSCHAR 01 CHARVAL CHAR(255)	CS_CHAR DBCHAR	1- to 255-byte fixed character type. TDSCHAR can be used to represent Japanese characters as well as alphabetic characters.
TDSDATETIME 01 DATTIME CHAR(8)	CS_DATETIME DBDATETIME	8-byte datetime datatype. The number of days since 1/1/1900, and the number of 300ths of a second since midnight.
TDSDATETIME4 01 DATTIM4 CHAR(4)	CS_DATETIME4 DBDATETIME4	4-byte datetime datatype. The number of days since 1/1/1900, and the number of minutes since midnight.
TDSFLT4 01 FLT4VAL BINARY FLOAT(21)	CS_REAL DBREAL	4-byte single precision type.
TDSFLT8 01 FLT8VAL BINARY FLOAT(53)	CS_FLOAT DBFLT8	8-byte double precision type.
TDSGRAPHIC 01 GRAPHVAL CHAR(254)	(Not applicable)	1- to-127-character fixed character type. Used at the mainframe only, to represent Japanese double-byte characters.

Open ServerConnect datatype and PL/1 data description	Client-Library/C and DB-Library datatypes	Datatype description
TDSIMAGE 01 IMAGEVAL CHAR(32000)	CS_IMAGE DBIMAGE	Sybase image datatype. A variable-length field that can hold from 0 to 2,147,483,647 bytes of binary data. This is a Sybase datatype and can be used only with column data being returned to Sybase clients.
TDSINT1 01 INT1VAL CHAR(1)	CS_TINYINT DBTINYINT	1-byte integer. Used with PL/1 only; not supported for COBOL.
TDSINT2 01 INT2VALFIXED BIN(15)	CS_SMALLINT DBSMALLINT	2-byte integer. Can be declared as numeric or character. When declared as numeric, it has a maximum value of 65,525.
TDSINT4 01 INT4VALFIXED BIN(31)	CS_INT DBINT	4-byte integer. Can be declared as numeric or character. When declared as numeric, it has a maximum value of 2,147,483,648.
TDSLONGVARBIN 01 LONGBINVALCHAR(32000)	CS_LONGBINARY	Long variable binary. The default maximum length for this datatype is 32K. Does not include the 2-byte (“LL”) length specification prefix. No translations are performed on this datatype.
TDSLONGVARCHAR 01 LONGCHARVALCHAR(32000)	CS_LONGCHAR	Long variable character type. The default maximum length for this datatype is 32K. Does not include the 2-byte (“LL”) length specification prefix.
TDSMONEY 01 MONEYVAL CHAR(8)	CS_MONEY DBMONEY	8-byte double precision type. The range of legal values for this datatype is: -$922,337,203,685,477.5807 to +$922,337,203,685,477.5807. *Note:* This datatype can be used with client data only.

Open ServerConnect datatype and PL/1 data description	Client-Library/C and DB-Library datatypes	Datatype description
TDSMONEY4 01 MONEY4VAL CHAR(4)	CS_MONEY4 DBMONEY4	4-byte double precision type. The range of legal values for this datatype is: -$214,748.3648 to +$214,748.3647 *Note:* This datatype can be used with client data only.
TDSNUMERIC 01 NUMVAL 05 NUM_PRECIS CHAR(1) 05 NUM_SCALE CHAR(1) 05 NUMARR CHAR(33)	CS_NUMERIC	Numeric type. Support for numbers with: *Precision* – the precision of the numeric value (1 to 77, default 18). *Scale* – scale of the numeric value (0 to 77, default 0). *Note:* This is a Sybase datatype. This datatype can be used for client data only.
TDS_PACKED_DECIMAL 01 PDECVAL FIXED DEC(15,2)	CS_PACKED370	IBM/370 packed decimal type. 31-byte precision is supported. *Note:* This is a mainframe datatype and can be used for mainframe data only.
TDS_SYBASE_DECIMAL 01 SDEC_VAL 05 NUM_PRECIS CHAR(1) 05 NUM_SCALE CHAR(1) 05 NUMARR CHAR(33)	CS_DECIMAL	Decimal type. Support for numbers with: *Precision* – the precision of the numeric value (1 to 77, default 18). *Scale* – scale of the numeric value (0 to 77, default 0). *Note:* This is a Sybase datatype. This datatype can be used for client data only.
TDSTEXT 01 TEXTVAL CHAR(32000)	CS_TEXT DBTEXT	Sybase text datatype. A variable-length field that can hold from 0 to 2,147,483,647 bytes of binary data. This is a Sybase datatype, and can be used only with column data being returned to Sybase clients.
TDSVARYBIN 01 VARGRAPH 05 VARBIN_LLFIXED BIN(15) 05 VARBIN_VAL CHAR(*n*)	CS_VARBINARY DBVARYBIN	1- to 255-byte variable binary type. The field length is stored in a 2-byte (“LL”) prefix, as in DB2. No translations are performed on this datatype.

Open ServerConnect datatype and PL/1 data description	Client-Library/C and DB-Library datatypes	Datatype description
TDSVARYCHAR 01 VARGRAPH 05 VARCHAR_LLFIXED BIN(15) 05 VARCHAR_VAL CHAR(*n*)	CS_VARCHAR DBVARYCHAR	1- to 255-byte variable character type. The field length is stored in a 2-byte (“LL”) prefix, as in DB2. TDSVARYCHAR can be used to represent Japanese characters as well as alphabetic characters.
TDSVARYGRAPHIC 01 VARGRAPH 05 VARGRAPH_LLFIXED BIN(15) 05 VARGRAPH_VAL CHAR(*n*)	(Not applicable)	1- to 255-byte graphic datatype. Used to represent Japanese double-byte characters. The field length is stored in a 2-byte (“LL”) prefix, as in DB2. *Note:* This is a mainframe datatype and can be used with mainframe data only.
TDSVOID	(Not applicable)	NULL or nonexistent parameter. Used to denote parameters omitted in stored procedures. For example, in the following RPC: exec rpc a, b, , d, e the missing c parameter is represented as TDSVOID. No translations are performed on this datatype.

Character datatypes

The following sections contain additional information about character datatypes.

TDSVARYCHAR

Always use TDSVARYCHAR rather than TDSVARCHAR. DBVARYCHAR and TDSVARYCHAR objects include a length specification that precedes the data, just like DB2 variable datatypes. The length specification occupies the initial two bytes of the field (in binary format) and is referred to in print as “LL”.

Note In Client-Library, the CS_VARCHAR datatype includes the “LL” length specifications and can be mapped to TDSVARYCHAR.

DB2 LONG VARCHAR datatypes

TDSLONGVARCHAR objects do not have the "LL" length specification. Programs using DB2 data can send DB2 LONG VARCHAR data as TDSVARYCHAR, TDSLONGVARCHAR, or TDSTEXT.

Converting to *TDSVARYCHAR*

If you use TDSVARYCHAR for LONG VARCHAR data, and if the text length is longer than 255 bytes, the data is either truncated or rejected. The truncation/rejection option is set at the TRS when it is started. The mainframe system programmer can override that option during customization.

Converting to *TDSLONGVARCHAR*

If you convert DB2 LONG VARCHAR data to TDSLONGVARCHAR, remember that this Open ServerConnect datatype does not have the "LL" length specification. Your program should point to the data portion of the declaration only.

Note If your client program is Open Client 10.0 or later, you can convert both columns and parameters to TDSLONGVARCHAR. Otherwise, you can use TDSLONGVARCHAR only with columns.

Converting to *TDSTEXT*

If you convert to TDSTEXT, the complete data string is sent, without truncation.

Binary and decimal datatypes

The following sections contain additional information about binary and decimal datatypes.

TDSVARYBIN

Use TDSVARYBIN rather than TDSVARBINARY. DBVARYBIN and TDSVARYBIN objects also include the "LL" length specification that precedes the data.

Note In Client-Library, the CS_VARBINARY datatype includes the "LL" length specifications and therefore can be mapped to TDSVARYBIN.

Converting Sybase decimal and numeric data

Use TDSNUMERIC and TDS_SYBASE_DECIMAL datatypes for Sybase Adaptive Server numeric and decimal data. These datatypes are defined as:

```
01 NUMDEC
    05 PRECISION CHAR X(1).
    05 SCALE     CHAR X(1).
    05 ARR       CHAR X(33).
```

In the preceding example, 1 byte is for precision, 1 byte is for scale, and 33 bytes are for the packed value.

You can use conversion between these datatypes and character data. Open ServerConnect also supports conversion between these datatypes and TDS_PACKED_DECIMAL (IBM packed decimal).

Converting packed decimal data

You can convert TDS_PACKED_DECIMAL to TDSNUMERIC, TDS_SYBASE_DECIMAL, character, float and money datatypes.

Converting packed decimal to character data

When converting TDS_PACKED_DECIMAL data to character datatypes, you must adjust the length of the result variable.

Use this formula to set the unpacked length:

```
Result Length =  (2 * Source Length) - 1.
```

When converting to character datatypes, automatic conversions may add a sign, a decimal point, and leading or trailing zeros. Allow one byte each for the sign and decimal point, and enough bytes to allow for the leading and trailing zeros.

When converting from packed decimal to character datatypes, Gateway-Library functions add zeros to the left of the decimal point for fractional values and to the right of the decimal point for integers. If no decimal point is present, one is added.

For all values, start with the defined length (precision):

- Add one byte for the sign:
  - If the sign is positive, Open ServerConnect adds a blank.
  - If the sign is negative, Open ServerConnect adds a minus sign.

For integer values:

- Add one byte for a decimal point
- Add one byte for a trailing zero

For fractional values: (n < 1 and > -1, precision = scale):

- Add one byte for a decimal point
- Add one byte for a leading zero

For non-integer values greater than 1:

- Add one byte for a decimal point

In the “Calculation” column in Table 2-10 on page 43, P=precision; s=sign; d=decimal point; z=zero(s). In the Character-Type Result column, b=blank.

Table 2-10: Examples of decimal-to-character conversions

Decimal value	Precision	Scale	Calculation P+s+d+z = result length	Character-type result	Result length
1	3	0	3+1+1+1 = 6	bbb1.0	6
123	3	0	3+1+1+1 = 6	b123.0	6
-123	3	0	3+1+1+1 = 6	-123.0	6
1.23	5	2	5+1+1+0 = 7	bbb1.23	7
.3	5	2	5+1+1+0 = 7	bbb0.30	7
-.2	5	2	5+1+1+0 = 7	bb0.20	7
123.45	5	2	5+1+1+0 = 7	b123.45	7
.123	3	3	3+1+1+1 = 6	b0.123	6
-.123	3	3	3+1+1+1 = 6	-0.123	6

For packed decimal-to-character conversions, the low-order digits of the character string are truncated. If the actual result is greater than the length of the destination, the low-order bytes are truncated.

For character-to-packed decimal conversions, the character string is scanned from left to right to determine precision and scale. The resulting packed decimal value contains the highest order digits that fit in the length specified by the destination length.

Packed decimal to numeric, decimal, float, money conversion

You can convert between IBM packed decimal and Sybase numeric, decimal, float or money datatypes.

You can also convert Sybase numeric, decimal, float or money to packed decimal. The result has the same scale as the source.

When converting from packed decimal to Sybase numeric, decimal, float or money, specify 35 as the destination length.

Sybase numeric or decimal to character data conversion

For numeric or decimal to character conversions, the precision and scale of the numeric data item are used to determine the output length of the character string. The source length should be the actual length of the numeric data item. The destination length should be precision + 2. If this length is less than the actual length of the result, TDS_OVERFLOW is returned.

For character to numeric or decimal conversions, the character string is scanned from left to right to determine precision and scale. You must specify the destination length as 35, or TDS_INVALID_LENGTH is returned. The numeric or data item contains the precision and scale as the first two bytes.

Graphic datatypes

Open ServerConnect programs can use graphic datatypes as well as character datatypes to process double-byte data. Workstation clients, however, use only character datatypes to represent characters; graphic datatypes are not used with the supported workstation character sets.

The length of mainframe graphic datatypes is the number of double-byte characters, whereas the length of character datatypes at both the mainframe and the workstation is the number of bytes. Therefore, when converting kanji from character to graphic datatypes, be aware that the length of a kanji string is twice as long for character datatypes as it is for graphic datatypes.

For a more detailed explanation of length considerations when converting Japanese characters, see Table 2-14 on page 61.

TDSVARYGRAPHIC

Use TDSVARYGRAPHIC rather than TDSVARGRAPHIC. TDSVARYGRAPHIC objects include the “LL” length specification that precedes the data.

DB2 LONG VARGRAPHIC datatypes

Programs using DB2 data can send DB2 LONG VARGRAPHIC data as TDSIMAGE.

Unsupported datatypes

If you attempt to send data with a datatype that is not supported by Open ServerConnect, the operation fails and returns an error.

Dynamic SQL support

Dynamic SQL allows a client application to execute SQL statements containing variables with values that are determined at run time. It is primarily useful for precompiler support. A client application prepares a dynamic SQL statement by associating a SQL statement containing placeholders with an identifier and sending the statement to an Open ServerConnect application so that the statement becomes a prepared statement.

When a client application is ready to execute a prepared statement, it defines values to substitute for the SQL statement placeholders and sends a command to execute the statement. These values become the command input parameters. After the statement executes the desired number of times, the client application deallocates the statement.

Dynamic SQL permits a client application to act interactively, passing different information at different times to the Open ServerConnect application, as it gets that information from the user. The Open ServerConnect application can then fill in the missing pieces in the SQL query with the data the user provides.

In Open ServerConnect, this process must occur as a long-running transaction. When a client issues a dynamic SQL command, Open ServerConnect indicates a TDS_DYNAMIC event through TDINFPGM or TDGETREQ. The server application retrieves the type of command through a TDYNAMIC call and then satisfies the client request.

Table 2-11 on page 46 defines the valid Open ServerConnect responses for various client requests.

Table 2-11: Valid dynamic SQL requests and responses

Client action	Open ServerConnect application response
Client issues a prepare command (TD_PREPARE)	1. Get operation type (TDS_GET) (TDYNAMIC) 2. Get statement ID length (TDS_GET) (TDYNAMIC) 3. Retrieve statement ID (TDS_GET) (TDYNAMIC) 4. Retrieve statement length (TDS_GET) (TDYNAMIC) 5. Retrieve statement (TDS_GET) (TDYNAMIC) 6. Send statement ID length (TDS_SET) (TDYNAMIC) 7. Send statement ID (TDS_SET) (TDYNAMIC) 8.Acknowledge request (TDS_SET) (TDYNAMIC) 9. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 10. Return a language, RPC, dynamic, or cursor request type (TDGETREQ)

Client action	Open ServerConnect application response
Client requests an input parameter description (CS_DESCRIBE_INPUT)	1. Get operation type (TDS_GET) (TDYNAMIC) 2. Get statement ID length (TDS_GET) (TDYNAMIC) 3. Retrieve statement ID (TDS_GET) (TDYNAMIC) 4. Send statement ID length (TDS_SET) (TDYNAMIC) 5. Send statement ID (TDS_SET) (TDYNAMIC) 6. Acknowledge request (TDS_SET) (TDYNAMIC) 7. Describe input parameters (TDESCRIB) 8. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 9. Get next request (TDGETREQ)

Client action	Open ServerConnect application response
Client requests an output parameter description (CS_DESCRIBE_OUTPUT)	1. Get operation type (TDS_GET) (TDYNAMIC) 2. Get statement ID length (TDS_GET) (TDYNAMIC) 3. Retrieve statement ID (TDS_GET) (TDYNAMIC 4. Describe output column(s) (TDESCRIB) 5. Send statement ID length (TDS_SET) (TDYNAMIC) 6. Send statement ID (TDS_SET) (TDYNAMIC) 7. Acknowledge request (TDS_SET) (TDYNAMIC) 8. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 9. Get next request (TDGETREQ)

Client action	Open ServerConnect application response
Client issues an execute request (TD_EXECUTE)	1. Get operation type (TDS_SET) (TDYNAMIC) 2. Get statement ID length (TDS_GET) (TDYNAMIC) 3. Retrieve statement ID (TDS_GET) (TDYNAMIC) 4. Retrieve number of parameters (TDNUMPRM) 5. Retrieve input parameter values (TDRCVPRM) 6. Send statement ID length (TDS_SET) (TDYNAMIC) 7. Send statement ID (TDS_SET) (TDYNAMIC) 8. Acknowledge request (TDS_SET) (TDYNAMIC) [application logic: execute client request] 9. Send result rows (TDSNDROW) 10. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 11. Return a language, RPC, dynamic, or cursor request type (TEGETREQ)

Client action	Open ServerConnect application response
Client issues an execute immediate request (TD_EXECUTE_IMMEDIATE)	1. Get operation type (TDS_GET) (TDYNAMIC) 2. Get statement ID (should be zero) (TDS_GET) (TDYNAMIC) 3. Retrieve statement length (TDS_GET) (TDYNAMIC) 4. Retrieve statement (TDS_GET) (TDYNAMIC) 5. Acknowledge request (TDS_SET) (TDYNAMIC) [application logic: execute client request] 6. Send result rows (TDSNDROW) 7. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 8.Return a language, RPC, dynamic, or cursor request type (TDGETREQ)

Client action	Open ServerConnect application response
Client issues a deallocation request (TD_DEALLOC)	1. Get operation type (TDS_GET) (TDYNAMIC) 2. Get statement ID length (TDS_GET) (TDYNAMIC) 3. Retrieve statement ID (TDS_GET) (TDYNAMIC) 4. Send statement ID length (TDS_SET) (TDYNAMIC) 5. Send statement ID (TDS_SET) (TDYNAMIC) 6. Acknowledge request (TDS_SET) (TDYNAMIC) 7. Send DONE packet (TDS_ENDREPLY) (TDSNDDON) 8. Return a language, RPC, dynamic, or cursor request type (TDGETREQ)

Events

Open ServerConnect responds to requests from clients. Some of these requests trigger an event in Open ServerConnect.

The API functions, TDINFPGM and TDGETREQ, return the request types in Table 2-12.

Table 2-12: Request types from TDINFPGM and TDGETREQ

Request type	If request is	Equivalent event
TDS_START_SQL	Language request	SRV_LANGUAGE
TDS_START_RPC	RPC request	SRV_RPC
TDS_CURSOR_EVENT	Cursor request	SRV_CURSOR
TDS_DYNAMIC_EVENT	Dynamic request	SRV_DYNAMIC

The login packet

The login packet can contain the following information:

- Name of TRS or mainframe listener.
- The client login information: name, name length, password, and, optionally, the originating application ID.
- The native language used at the client workstation.
- The character set used by the client. This can be a standard character set name (such as iso-1) or the name of a customer-defined character set.
- The type of request (language, RPC, cursor or dynamic).

The client program sets this information in a login packet and sends it to the mainframe through TRS or mainframe listener. The login packet is passed when the transaction starts.

Open ServerConnect calls retrieve and use information from the login packet as necessary. An Open ServerConnect program can examine some of the data in the login packet by calling TDGETUSR. See TDGETUSR on page 112 for more details.

Long-running transactions

In the standard (short) transaction model, a mainframe transaction ends as soon as it finishes sending results to a single client request.
A long-running transaction does not end the transaction when all results are sent, but remains active, ready to accept additional requests.

Note Long-running transactions are supported with CICS, MVS, and with the IMS TM explicit API, but not with the IMS TM implicit API. To simulate a long-running transaction in the IMS TM implicit API, you must define the transaction as a WFI (wait-for-input) transaction in the TRANSACT macro.

Long-running transactions begin like transactions that process single client requests, but, instead of closing the connection after returning results, they switch from SEND to RECEIVE state, ready to accept subsequent requests. Because a transaction can call TDACCEPT only once, it calls TDGETREQ to process subsequent client requests. TDGETREQ also returns the type of request received.

The values assigned to TDSNDDON arguments determine the type of transaction.

- For short transactions:
  - Set *STATUS* to TDS_DONE_FINAL.
  - Set *CONN_OPTIONS* to TDS_ENDRPC. This closes the connection and ends the conversation.
- For long-running transactions (when preparing to accept another request).
  - Set *STATUS* to TDS_DONE_FINAL.
  - Set *CONN_OPTIONS* to TDS_ENDREPLY. This switches the communication state to RECEIVE. The TDS_ENDREPLY option indicates that the host is expecting a subsequent communication from the client.
- For IMS TM WFI transactions:
  - Set *STATUS* to TDS_DONE_FINAL.
  - Set *CONN_OPTIONS* to TDS_ENDRPC. The transaction can call TDGETREQ to accept another client request.

A transaction can determine the communication state by calling TDSTATUS. TDSTATUS returns TDS_SEND or TDS_RECEIVE while the transaction is running and TDS_RESET when a CICS, MVS, or IMS TM explicit transaction ends.

Note For IMS TM Users, IMS TM WFI transactions can accept additional RPCs after receiving TDS_RESET. See TDSTATUS on page 235 for details.

Calls in a long-running transaction

The pattern of calls in a long-running transaction is as follows:

The first client request

A long-running transaction processes the first client request the same way any transaction processes a client request.

For the first client request, a long-running transaction:

1 Calls TDACCEPT to accept the request.

2 Uses TDINFPGM to determine the type of request received. This will be an RPC or a language, cursor, or dynamic request.

3 Processes the request and returns results.

4 Calls TDSNDDON with *CONN_OPTIONS* set to TDS_ENDREPLY, which puts the mainframe in RECEIVE state, ready to receive another request.

Subsequent client requests

For subsequent client requests, a long-running transaction:

1 Calls TDGETREQ to accept each subsequent request and determine whether it is an RPC or a language, cursor, or dynamic request.

2 Processes the request and returns results.

3 Calls TDSNDDON with *CONN_OPTIONS* set to TDS_ENDREPLY, which puts the mainframe in RECEIVE state, ready to receive another request.

The final client request

A long-running transaction must free up all resources after it accepts and processes the last client request. It treats the new request as any other subsequent client request, then calls TDTERM to end the transaction.

For the final client request, a long-running transaction:

1 Calls TDGETREQ, with *WAIT_OPTION* set to TDS_FALSE, to:

- Accept the final request, if one is present, or
- End the transaction, if no request is pending.

2 Processes the request and returns results.

3 Calls TDSNDDON with *CONN_OPTIONS* set to TDS_ENDRPC, which ends the transaction.

4 Calls TDFREE.

5 Calls TDTERM to free up all resources.

Note TDTERM is required for IMS TM and MVS. It is optional but recommended for CICS.

Refer to “Processing a long-running transaction” on page 13 to see the skeleton of a basic long-running transaction.

Mixed-mode applications

Mixed-mode applications are application programs that use both Gateway-Library and Client-Library functions. In other words, they act as both server and client.

One example of a mixed-mode application is a transaction that accepts requests from a remote client, and then sends requests containing the client data to a remote server. When the transaction receives results from that server, it returns them to the remote client.

Rules for writing mixed-mode applications

Follow these rules when writing mixed-mode applications:

- The first Open ServerConnect or Client-Library call must be TDINIT.
- Call TDACCEPT before calling any Client-Library functions.

  TDACCEPT allocates the handle for the connection to the remote client, reads in client login information, and does the necessary translations.

  We recommend using Gateway-Library "receive" functions (TDRCVPRM, TDRCVSQL) to retrieve client data before calling any Client-Library functions when you use different levels of Open ClientConnect and Open ServerConnect. Otherwise, Client-Library calls will not abend but can get out of synchronization.
- After the final results are sent to the remote client, use TDFREE and TDTERM to end the transaction.

A sample mixed-mode application, SYCTSAX4, is in Appendix F, "Sample Mixed-Mode Application."

Native languages

Open ServerConnect can accept and process requests in a variety of native languages. The following native languages are available from Sybase for Open ServerConnect:

- U.S. English
- French
- German
- Japanese

Your system programmer can customize Open ServerConnect at your site to add additional native languages.

An Open ServerConnect program can query the native language with TDGETUSR.

Processing Japanese client requests

Note The Japanese Conversion Module (JCM) is available for CICS only. If you are not using the JCM, you can skip this section.

The Japanese Conversion Module

Open ServerConnect can accept and process client requests written in Japanese if you have the JCM installed. The JCM is provided on a separate tape. It does the workstation-to-mainframe-to-workstation translations necessary to process requests containing Japanese characters.

Customization

The Open ServerConnect environment must be customized to process Japanese requests. A system programmer customizes your environment when Open ServerConnect is installed. Open ServerConnect loads the customization module when TDINIT is called.

Customization information includes client login information from the client login packet that TRS forwards to the mainframe along with the client request. Among the client information contained in the login packet is the name of the client character set. See “The login packet” on page 52 for details.

The following options are set during customization:

- The native language used at the mainframe
- DBCSs; whether double-byte kanji characters are used
- Information about single-byte character sets

  The use of this option depends on whether DBCS is used:

  - If double-byte characters are used, this option indicates whether single-byte characters are treated as hankaku katakana or as lowercase alphabetic characters. The default, as shipped, is hankaku katakana.

- If double-byte characters are not used, this option names the default (single-byte) character set. In the current version, the default character set is iso-1.

If the native language is Japanese, TDINIT loads the JCM.

An Open ServerConnect program can retrieve customization information with the function TDGETUSR.

How the JCM works

Once the JCM is loaded, it gets control whenever an Open ServerConnect program receives a client request containing TDSCHAR or TDSVARYCHAR data. TDSCHAR and TDSVARYCHAR are the datatypes used to represent Japanese characters in workstation character sets. The JCM converts the workstation Japanese characters to the character set used on the mainframe. Once mainframe processing is completed, the JCM converts results back to the original workstation character set before returning them to the client.

The translate tables

The JCM uses translate tables to convert workstation characters to mainframe characters.

When an Open ServerConnect program receives a client request in Japanese that contains character datatypes, it gives control to the JCM. The JCM looks up the client character set in the translate tables.

- If the JCM finds a translate table for the client character set, the JCM converts the data and names into the equivalent mainframe characters. After processing is complete, the JCM converts results back to the workstation characters before returning results to the client.
- If the client does not specify a character set in the login packet, or if the JCM cannot find a translate table for the client character set, the program fails, and Open ServerConnect sends the client an error message.

Japanese character sets

Different brands of workstations use different character sets to represent double-byte characters. See “Character sets” on page 17 to learn what single-byte and double-byte character sets are supported on the workstation and at the mainframe.

Differences among Japanese character sets

Each character set used to handle Japanese characters has its own way of representing kanji or hankaku katakana characters and specifying lengths for Japanese character strings. While most of the differences are handled by the JCM, you need to understand a few of these differences in order to specify field lengths correctly. These differences are discussed in this section.

Datatypes used with Japanese characters

The following datatypes can be used with Japanese characters at the workstation:

- TDSCHAR
- TDSVARYCHAR

The following datatypes can be used with Japanese characters at the mainframe:

- TDSCHAR
- TDSVARYCHAR
- TDSGRAPHIC
- TDSVARYGRAPHIC

  Graphic datatypes are used with double-byte characters only.

Kanji datatypes

Kanji characters always occupy 2 bytes.

Hankaku Katakana datatypes

Hankaku katakana characters are always represented as single-byte character-type data with datatypes of TDSCHAR or TDSVARYCHAR.

Kanji string lengths

Kanji characters are represented as character-type data at the workstation, and as either character-type or graphic-type data at the mainframe. The length of a Japanese character string depends on which workstation is being used and whether the datatype is graphic or character.

Some character sets use a special indicator or code in character-type strings to announce that the following series of characters are double-byte characters. With kanji, this indicator is called a Shift Out (SO) code. An SO code marks the beginning of a double-byte kanji string. The end of the kanji string is marked by a Shift In (SI) code.

When setting field lengths for Japanese character strings, you must include room for these SO/SI codes.

When sending data from a mainframe to a workstation, you can replace SO/SI codes with blanks by calling the Gateway-Library function TDSETSOI before receiving or sending data.

Graphic datatypes do not use SO/SI codes.

Warning! When receiving data from a workstation character set that does not use SO/SI codes, IBM_Kanji always inserts the SO/SI codes at the beginning and end of double-byte character strings. If the field length specification does not take this into account, and the length is just long enough for the data itself, some of the data is lost.
If a field contains mixed single-byte and double-byte data in more than one kanji string, an SO/SI pair exists for each kanji string.

At the mainframe, the length of graphic-type strings is counted in double-byte (16-bit) characters. Thus, a string of 10 kanji characters has a length of 10.

At the workstation, the length of kanji character strings is counted in bytes. Thus, a string of 10 kanji characters has a length of 20.

Hankaku Katakana string lengths

The length of a hankaku katakana string is always represented in bytes, at both the workstation and the mainframe. A hankaku katakana character occupies one byte, except in eucjis.

The eucjis hankaku katakana character set uses an indicator (SS2) in character-type strings to announce that the next byte is occupied by a hankaku katakana. The SS2 indicator occupies one byte, and the hankaku katakana itself occupies one byte. As a result, the total length of each eucjis hankaku katakana character is two bytes.

Summary of datatypes used with Japanese characters

The following datatypes listed in Table 2-13 are used with Japanese characters.

Table 2-13: Datatypes used with Japanese characters

Datatype	Used with	Uses SO/SI or SS2	Length measures
TDSCHAR TDSVARYCHAR	DBCS and SBCS. At the workstation and at the mainframe.	*IBM Kanji*: Uses SO/SI with double-byte characters. *EUC-JIS*: Uses SS2 with hankaku katakana.	For all character sets: Number of bytes. Maximum length for TDSCHAR and TDSVARYCHAR is 255.
TDSGRAPHIC TDSVARYGRAPHIC	DBCS only. At mainframe only.	No.	Number of characters. Maximum length is 127.

Length considerations

When converting from a workstation Japanese character set to a mainframe Japanese character set, you frequently need to adjust the length.
The adjustment depends on which character sets, datatypes, and language are being used.

In this section:

- Descriptions of eucjis data also apply to deckanji, with the exception that deckanji does not include hankaku katakana.
- Open ServerConnect character datatypes are TDSCHAR and TDSVARYCHAR.
- Open ServerConnect graphic datatypes are TDSGRAPHIC and TDSVARYGRAPHIC.
- Open ServerConnect datatypes with "VARY" in the name have a two-byte length ("LL") specification at the beginning of each data field. Do not count these "LL" bytes when calculating the length of the field.

Character set length requirements

Table 2-14 describes how Japanese characters are represented in supported character sets, and how their lengths are affected.

Table 2-14: Length requirements in Japanese character sets

Character set	SBCS or DBCS	Datatype	Length considerations	Example
EUC-JIS	DBCS (hankaku katakana)	character	Each 1-byte hankaku katakana character is preceded by a 1-byte SS2 indicator. As a result, each eucjis hankaku katakana character has a length of 2: the SS2 indicator and the hankaku katakana itself.	A string of 4 hankaku katakana occupies 8 bytes and has a length of 8.
EUC-JIS	DBCS (kanji)	character	Each kanji character is 2 bytes long and has a length of 2. Kanji and single-byte alphabetic characters can be mixed. When converting mixed strings from IBM Kanji to workstation kanji, double the length to be safe.	A string of 4 kanji occupies 8 bytes and has a length of 8.
Shift-JIS	SBCS (hankaku katakana)	character	Each hankaku katakana character is 1 byte long and has a length of 1. Shift-JIS hankaku katakana does not use SS2 indicators.	A string of 4 hankaku katakana occupies 4 bytes and has a length of 4.
Shift-JIS	DBCS (kanji)	character	Each kanji character is 2 bytes long and has a length of 2. Kanji and single-byte alphabetic characters can be mixed. When converting mixed strings from IBM Kanji to workstation kanji, double the length to be safe.	A string of 4 kanji occupies 8 bytes and has a length of 8.
IBM Kanji kanji	DBCS	character	Each kanji character is 2 bytes long and has a length of 2. Each kanji string is preceded by a Shift Out indicator and followed by a Shift In indicator, adding two to the length of each kanji string. Kanji and single-byte alphabetic characters can be mixed. When converting mixed strings from IBM Kanji to workstation kanji, double the length to be safe.	A string of 4 kanji occupies 10 bytes and has a length of 10. (8 bytes for the data and 2 bytes for the SO/SI codes)

Character set	SBCS or DBCS	Datatype	Length considerations	Example
IBM Kanji kanji	DBCS	graphic	Each kanji character is a double-byte character and has a length of 1. There are no SO/SI indicators with graphic data.	A string of 4 kanji occupies 8 bytes and has a length of 4.
IBM Kanji hankaku katakana	SBCS	character	Each hankaku katakana character is 1 byte long and has a length of 1. IBM Kanji hankaku katakana does not use SS2 indicators.	A string of 4 hankaku katakana occupies 4 bytes and has a length of 4.

Examples of length settings in conversions

Table 2-15 illustrates length adjustments required for some workstation-to-mainframe Japanese character set conversions.

Table 2-15: Length-settings in Japanese character set conversions

Source character set	Source datatypes	Source length	Target character set	Target datatypes	Target length
EUCJIS hankaku katakana	character	8	IBM Kanji hankaku katakana	character	4
EUCJIS kanji	character	8	IBM Kanji kanji	character	10
EUCJIS kanji	character	8	IBM Kanji kanji	graphic	4
Shift-JIS hankaku katakana	character	4	IBM Kanji hankaku katakana	character	4
Shift-JIS kanji	character	8	IBM Kanji kanji	character	10
Shift-JIS kanji	character	8	IBM Kanji kanji	graphic	4
IBM Kanji hankaku katakana	character	4	EUCJIS hankaku katakana	character	8
IBM Kanji hankaku katakana	character	4	Shift-JIS hankaku katakana	character	4
IBM Kanji kanji	character	10	EUCJIS kanji	character	8
IBM Kanji kanji	character	10	Shift-JIS kanji	character	8

Source character set	Source datatypes	Source length	Target character set	Target datatypes	Target length
IBM Kanji kanji	graphic	4	EUCJIS kanji	character	8
IBM Kanji kanji	graphic	4	Shift-JIS kanji	character	8

Lengths in conversions

Because differences among Japanese character sets can result in longer and shorter lengths after conversion, Gateway-Library includes the TDSETSOI function that specifies padding or stripping the SO/SI indicators.

When converting from a character set that uses SO/SI indicators to one that does not (for example, converting CHAR data from IBM Kanji to Shift-JIS kanji), you can use TDSETSOI to specify whether the SO/SI indicators are stripped or whether they are replaced with embedded blanks. When replaced with embedded blanks, the length does not change. When stripped, the length is reduced by two bytes for each kanji string.

If no strip option is set, the JCM automatically strips SO/SI indicators.

When TDSETSOI replaces SO/SI indicators with blanks, the blanks are positioned at the end of the field. For example, in an IBM Kanji CHAR field that contains four kanji, the first byte contains the SO indicator, and the tenth byte contains the SI indicator. After conversion to Shift-JIS kanji, the first eight bytes are occupied by kanji, and the blanks occupy bytes nine and ten.

By judicious use of TDSETSOI, you can minimize the length changes and calculations needed in Open ServerConnect programs. See TDSETSOI on page 203 for details.

CHAPTER 3

Functions

This chapter describes the Gateway-Library functions that are included with your Open ServerConnect software. Table 3-1 lists the functions and provides a brief description of each. Following Table 3-1 is general information about functions and then a detailed description of each listed one.

List of functions

Table 3-1 describes the Gateway-Library functions.

Table 3-1: List of Gateway-Library functions

Function	Description
TDACCEPT	Accepts an incoming request from a remote client.
TDCONVRT	Converts a mainframe datatype to a datatype that can be used by an Open Client DB-Library or Client-Library application.
TDCURPRO	Gets or sets information about a cursor.
TDESCRIB	Describes a row column and binds its associated host program variable.
TDFREE	Frees up the TDPROC structure for the connection.
TDGETREQ	Accepts the next RPC or language, cursor, or dynamic request in a long-running transaction and returns the transaction ID of the associated mainframe transaction.
TDGETSOI	Determines what Shift Out/Shift In (SO/SI) processing options are set for a column or parameter. (Used with double-byte character sets.)
TDGETUSR	Retrieves client login and mainframe customization information.
TDINFACT	Retrieves information about global accounting.
TDINFBCD	Retrieves the length and the number of decimal places for a specified packed decimal column or parameter.
TDINFLOG	Returns information about the trace settings currently in effect for the trace log.
TDINFPGM	Returns information about the currently running transaction.
TDINFPRM	Retrieves information about the specified parameter.
TDINFRPC	Returns information about the client RPC that requested the current transaction.

Function	Description
TDINFSPT	Indicates whether tracing is on or off for a specified transaction, and returns the transaction ID.
TDINFUDT	Returns information about the user-defined datatype associated with a column.
TDINIT	Initializes the Gateway-Library environment.
TDLOCPRM	Returns the ID number of a parameter based on its parameter name.
TDLSTSPT	Lists all transactions for which tracing is enabled.
TDNUMPRM	Returns the total number of parameters that came with the current remote procedure call, or cursor or dynamic request.
TDRCVPRM	Receives a parameter from a remote client.
TDRCVSQL	Receives a SQL statement string from a remote client.
TDRESULT	Describes the communication received from the client.
TDSETACT	Turns system-wide accounting for Gateway-Library on or off. Also used to rename the accounting log under CICS.
TDSETBCD	Specifies the length and number of decimal places for a given column.
TDSETLEN	Sets the column length for a variable-length field before sending it to a client.
TDSETLOG	Turns system-wide tracing options for the Gateway-Library functions on or off. Also used to rename the trace log under CICS.
TDSETPRM	Specifies the length and address of a return parameter.
TDSETPT	Specifies the type of IMS TM transaction being used.
TDSETSOI	Sets the Shift Out/Shift In (SO/SI) processing options for a column or parameter. (Used with double-byte character sets.
TDSETSPT	Turns tracing on or off for a specified transaction.
TDSETUDT	Specifies the TDS datatype for a given column.
TDSNDDON	Sends return parameter information back to the client. Tells the client that all results have been sent, and retains or terminates the connection between the client and server.
TDSNDMSG	Sends an error or informational message to a client.
TDSNDROW	Sends a row of data back to the requesting client.
TDSQLLEN	Gets the length of a SQL statement string received from a remote client.
TDSTATUS	Retrieves status information about the client/server connection.
TDYNAMIC	Read or respond to a client dynamic SQL command.
TDWRTLOG	Writes a user-created message or a system entry to the trace log.

General information about functions

Each entry on the following pages includes a functional description, the syntax (including the datatype of each argument), an example, possible return codes, and a list of related topics. All arguments of each function are required unless designated as optional.

Most examples in this chapter are taken from the sample programs in the appendixes. These programs show how individual functions are coded in context. Refer to these appendixes to learn how to set up your work areas and to see examples of complete programs.

Your application must include a set of constants supplied with Open ServerConnect. These are standard argument options and return values for the PL/1 Gateway-Library interface. To include these constants in a PL/1 program, include the copybook SYGWPLI in the program.

When you use Gateway-Library functions, be aware of the following information:

- For most Gateway-Library functions, the return codes for Gateway-Library functions are stored in a *RETCODE* argument. Where this argument exists, it is always the second argument for a function.
- The *TDPROC* structure is established by the TDACCEPT function. *TDPROC* is a required argument of all subsequent functions that use the same connection (except tracing and accounting functions). In most cases, *TDPROC* is the first argument.
- The maximum length allowed for names is 30 bytes.

Note For numeric and alphabetic lists of all return codes, with descriptions, see Mainframe Connect Client Option and Server Option *Messages and Codes*.

TDACCEPT

Description

Accepts a request from a remote client. This function returns the handle for the SNA or TCP/IP conversation in the *TDPROC* program variable.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC               PTR,
01 RETCODE               FIXED BIN(31),
01 IHANDLE              PTR,
01 ACCEPT_CONNECTION_NAME CHAR(08) INIT('  '),
01 ERROR_SUBCODE         FIXED BIN(31);

CALL TDACCEPT (TDPROC,RETCODE,IHANDLE,
          ACCEPT_CONNECTION_NAME, ERROR_SUBCODE);
```

Parameters

TDPROC

(O) Handle for this client/server connection. All subsequent server functions using this connection must specify this same value in the *TDPROC* argument. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-2 on page 68.

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

ACCEPT_CONNECTION_NAME

(I) Leave blank. CICS and IMS TM get this information elsewhere.

ERROR_SUBCODE

(O) Detailed error information. Provides additional information about the cause of failure when TDACCEPT returns a return code other than TDS_OK. For a list of error subcodes, see Mainframe Connect Client Option and Server Option *Messages and Codes*.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-2.

Table 3-2: TDACCEPT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.

Return value	Meaning
TDS_CHARSET_NOTLOADED (-261)	Gateway-Library found the DBCS specified by the client, but the corresponding double-byte module was not loaded at the mainframe. This code is retuned to TDACCEPT when a client specifies a DBCS (for example, Shift-JIS) for which the associated translate module was not loaded or defined to the mainframe system. If the TP system is CICS, this can mean that the translate module was not defined in RDO (or to the PPT table), or that it is not present in the LOADLIB.
TDS_CHARSETSRV_NOT_SBCS (-264)	The client character set was not found; DBCS specified as default. This code represents two problems: • The character set named in the client login packet was not found in the table of character set names. This may indicate that the client did not specify the character set correctly (for example, the -J option in isql or the DBSETLCHARSET value in a DB-Library program is invalid). • Open ServerConnect was customized to process single-byte character sets, but the default character set is double-byte. This usually indicates that the customization settings are incorrect for kanji support.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_DBCS_CHARSET_NOTFOUND (-263)	Gateway-Library could not find the DBCS specified in the client login packet. This usually indicates that the client request specified an invalid character set in, for example, the -J option in isql or the DBSETLCHARSET value in a DB-Library program.
TDS_DEFAULT_CHARSET_NOTFOUND (-262)	The client login packet did not specify a character set or the specified client character set could not be found, and Gateway-Library did not find the default. This code is returned for SBCSs only.
TDS_GWLIB_UNAVAILABLE (-15)	Could not load SYGWCICS (the Gateway-Library phase).
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.

Return value	Meaning
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_SOS (-257)	Memory shortage. The host subsystem was unable to allocate enough memory for the control block that Gateway-Library was trying to create. The operation failed.
TDS_USING_DEFAULT_CHARSETSRV (10)	Gateway-Library using default character set. The client login packet did not specify a character set, or Gateway-Library could not find the specified single-byte character set, so it used the default character set specified during customization. This is an informational message.

Examples

Example 1

The following code fragment illustrates the use of TDINIT, TDACCEPT, TDSNDDON, TDFREE, and TDTERM at the beginning and end of a Gateway-Library program. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*-----------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-----------------------------------------------------------------*/

/*      ---------------------------------------------------------*/
/*      reset db2 error handlers                                 */
/*      ---------------------------------------------------------*/
       EXEC SQL WHENEVER SQLWARNING CONTINUE;
       EXEC SQL WHENEVER SQLERROR   CONTINUE;
       EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/*      ---------------------------------------------------------*/
/*      establish gateway environment                            */
/*      ---------------------------------------------------------*/
       CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);

/*      ---------------------------------------------------------*/
/*      accept client request                                    */
/*      ---------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME,
                      SNA_SUBC);
```

```
    /*-------------------------------------------------------------------*/
    /*  (BODY OF PROGRAM)                                                */
    /*-------------------------------------------------------------------*/
          GO TO END_PROGRAM;
    /*-------------------------------------------------------------------*/
    END_PROGRAM:
    /*-------------------------------------------------------------------*/
          IF SEND_DONE = SEND_DONE_OK THEN
              WRK_DONE_STATUS = TDS_DONE_COUNT;

          ELSE
          DO;
              WRK_DONE_STATUS  = TDS_DONE_ERROR;
              PARM_RETURN_ROWS = 0;
          END;

          CALL TDSNDDON (GWL_PROC, GWL_RC,
                         WRK_DONE_STATUS,
                         PARM_RETURN_ROWS,
                         TDS_ZERO,
                         TDS_ENDRPC);

          CALL TDFREE (GWL_PROC, GWL_RC);
          EXEC CICS RETURN;

  END SYSAMP1;
```

Example 2

The following code fragment illustrates the use of TDINIT, TDSETPT, and TDACCEPT at the beginning of a Gateway-Library program. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Explicit).”

```
/*      ------------------------------------------------------------*/
/*       establish gateway environment                             */
/*      ------------------------------------------------------------*/
         CALL TDINIT (P_PCBTERM, GWL_RC, GWL_INIT_HANDLE);
   [check return code]
/*      ------------------------------------------------------------*/
/*        set program type to EXPL                                  */
/*      ------------------------------------------------------------*/
          GWL_PROG_TYPE = 'EXPL';

          CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                        GWL_SPA_PTR, TDS_NULL, TDS_NULL);
   [check return code]
```

```
/*      ------------------------------------------------------------*/
/*      accept client request                                       */
/*      ------------------------------------------------------------*/
          CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                         SNA_CONNECTION_NAME,
                         SNA_SUBC);
    [check return code]
```

Usage

TDACCEPT

A mainframe server application uses this function to accept RPCs and language requests from a remote client. It reads in login information and does all necessary translations, including workstation-to-mainframe character set conversions for all supported national languages. This function is required in all Open ServerConnect programs.

- TDACCEPT retrieves login information from the login packet sent with the client request. The login packet contains information needed by the mainframe program, including the following:
  - The client login name and password.
  - The national language and character set used by the client.
  - The type of request (language, RPC, cursor or dynamic).
    If the request is an RPC, the RPC name. If the request is a cursor, the cursor name.
  - The name of the TRS sending the request (in a three-tier environment). The name of the Open Server from the interfaces file (in a two-tier environment).

  Your program can retrieve login information with TDGETUSR.
- TDACCEPT returns the handle for the conversation initiated by this client request.

  Note This book uses the term *conversation* to refer to active connections for both SNA and TCP/IP.
- A successful TDACCEPT puts the server application in RECEIVE state. The server application can then call TDRCVSQL, TDYNAMIC, TDRCVPRM or TDCURPRO to retrieve incoming SQL text,
  RPC parameters or cursor information. See “Communication states” on page 19 for a discussion of SEND and RECEIVE states.

- TDACCEPT returns standard communication subcodes. These codes are listed in Mainframe Connect Client Option and Server Option *Messages and Codes*.
- Only one TDACCEPT can be in an Open ServerConnect program. If this is a long-running transaction, use TDACCEPT to accept the first client request and TDGETREQ to accept subsequent requests.

Character set conversion

After Gateway-Library accepts the client request, it converts the request into a from understood by the mainframe. Roman characters are converted from ASCII to EBCDIC. Japanese characters are converted to IBM-Kanji.

Gateway-Library uses translation tables to do these conversions. Single-byte translation tables can be customized locally. The Japanese Conversion Module has its own set of conversion tables.

Customization

The Open ServerConnect environment is customized at the customer site. During customization, you define the type of requests that Gateway-Library will process. Customized items related to international applications include:

- The national language used at the mainframe.
- The DBCS support flag. This determines whether or not DBCSs are supported.
  - If DBCSs are supported, then SBCSs are customized as to whether they are treated as lowercase roman letters, or as Japanese hankaku katakana characters.
- The default character set, when the client character set is single-byte.

Login packet

When TEDACCEPT retrieves the client character set from the login packet, it looks up that character set in a table of supported character set names. If it finds a match in that table, it uses the associated translation table or conversion module to convert the request to mainframe characters.

If no character set is specified in the login packet, or if Gateway-Library cannot find a match for the specified client character set, the action taken by TDACCEPT depends on whether or not a DBCS was specified during customization.

When the character set is single-byte:

- Gateway-Library uses the default character set defined during customization, and TDACCEPT returns TDS_OK.

- If Gateway-Library cannot find the default character set in the character set table, TDACCEPT fails and returns TD-DEFAULT-CHARSET-NOTFOUND.
- If Gateway-Library find the default character set, but it is a DBCS, TDACCEPT fails, returning TDS-CHARSETSRV-NOT-SBCS.

When the character set is double-byte, and:

- The login packet does not specify a character set or specifies one that Gateway-Library cannot match, then:
  - TDACCEPT fails and returns TDS-DBCS-CHARSET-NOTFOUND.
- Gateway-Library finds the client character set, but the corresponding conversion module (for example, the JCM) was not loaded, TDACCEPT fails and returns TDS-CHARSET-NOTLOADED or TDS-CONTROL-NOT-LOADED.

For Japanese users

Japanese requests are processed by the Japanese Conversion Module (JCM), a separate tape that provides Japanese language support for Open ServerConnect. The JCM must be installed and defined to your mainframe system before Gateway-Library can process client requests written in Japanese.

Loading the JCM

Within a Gateway-Library program, TDINIT loads the JCM. If it cannot load that module, TDINIT does not return an error code. However, when a client request specifies a DBCS in the login packet, TDACCEPT returns TDS-CHARSET-NOTLOADED.

If your program uses the JCM, TDACCEPT converts the name of each parameter to the character set used at the mainframe.

See also

Related functions

- TDFREE on page 99
- TDINIT on page 148

Related topics

- “Character sets” on page 17
- “The login packet” on page 52
- “Processing Japanese client requests” on page 56
- “Customization” on page 35

Related documents

- Mainframe Connect Client Option and Server Option *Messages and Codes*

TDCONVRT

Description

Converts the data in a variable from a mainframe datatype to a datatype that can be used by an Open Client program.

Note TDCONVRT converts SBCS. Do not use it with DBCS.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC            PTR,
01 RETCODE           FIXED BIN(31),
01 NUM_DECIMAL_PLACES FIXED BIN(31),
01 SOURCE_TYPE       FIXED BIN(31),
01 SOURCE_LENGTH     FIXED BIN(31),
01 SOURCE_VARIABLE   CHAR(n),
01 RESULT_TYPE       FIXED BIN(31),
01 RESULT_LENGTH     FIXED BIN(31),
01 RESULT_VARIABLE   CHAR(n),
01 OUTLEN            BIN(31);

CALL TDCONVRT (TDPROC, RETCODE, NUM_DECIMAL_PLACES,
          SOURCE_TYPE, SOURCE_LENGTH,
          SOURCE_VARIABLE, RESULT_TYPE,
          RESULT_LENGTH, RESULT_VARIABLE,
          OUTLEN);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-3 on page 76.

NUM_DECIMAL_PLACES

(I) Number of digits after the decimal point (scale) in the *SOURCE_VARIABLE*. This value must not be a negative number. When converting packed decimal to or from numeric or Sybase-decimal, or when converting packed decimal, numeric, or Sybase decimal to or from character format, TDCONVRT uses this information to ensure that the decimal point is correctly placed. For all other datatypes, it ignores this argument.

SOURCE_TYPE

(I) Datatype of the *SOURCE_VARIABLE*.

SOURCE_LENGTH

(I) Actual length of the *SOURCE_VARIABLE*. This value must not be a negative number. For TDSVARYCHAR or TDSVARYBIN this value does not include two bytes for "LL" specifications. For Sybase numeric or decimal, it is actual length not a maximum length (35).

SOURCE_VARIABLE

(I) Host program variable that contains the data to be converted. This is the variable described in the previous two arguments.

RESULT_TYPE

(I) DB-Library or Client-Library datatype of the *RESULT_VARIABLE*.

RESULT_LENGTH

(I) Actual length of the *RESULT_VARIABLE*. This value must be greater than zero and must not be a negative number. For fixed-length datatypes, this argument is ignored. Always use 35 as a result length for numeric and Sybase decimal data.

RESULT_VARIABLE

(O) Variable that contains the converted data. This is the variable described in the previous two arguments.

OUTLEN

(O) Optional, returns actual length for numeric or Sybase decimal result.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-3.

Table 3-3: TDCONVRT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.

Return value	Meaning
TDS_DATE_CONVERSION_ERROR (-23)	Error in conversion of datetime data. This can be a result of trying to convert short datetime (TDSDATETIME4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short datetime datatype.
TDS_DECIMAL_CONVERSION_ERROR (-24)	Error in conversion of packed decimal data.
TDS_FLOAT_CONVERSION_ERROR (-21)	Error in conversion of float values.
TDS_INVALID_DATA_CONVERSION (-172)	Incompatible datatypes. The source datatype cannot be converted into the requested result datatype.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *RESULT_LENGTH* argument is too short.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.
TDS_MONEY_CONVERSION_ERROR (-22)	Error in conversion of TDSMONEY-type data. This can be a result of trying to convert to short money (TDSMONEY4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short money datatype.
TDS_TRUNCATION_ERROR (-20)	Error occurred in truncation of data value.

Examples

The following code fragment uses TDCONVRT to convert data from the DB2 datatypes to DB-Library datatypes. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*-------------------------------------------------------------------*/
SETUP_REPLY_COLUMNS:
/*-------------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_FNM);
      DB_COLUMN_NAME_HV_PTR  = ADDR(CN_FNM);
      WRKLEN1                = STG(EMPLOYEE_FNM)-2;
      WRKLEN2                = STG(CN_FNM);
      DB_HOST_TYPE           = TDSVARYCHAR;
      DB_CLIENT_TYPE         = TDSVARYCHAR;
      CALL DESCRIBE_COLUMN;
/*      ---------------------------------------------------------*/
/*      Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR) */
/*      to DBCHAR.                                                 */
/*      ---------------------------------------------------------*/
```

```
        DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_LNM);
        DB_COLUMN_NAME_HV_PTR  = ADDR(CN_LNM);
        WRKLEN1                = STG(EMPLOYEE_LNM)-2;
        WRKLEN2                = STG(CN_LNM);
        DB_HOST_TYPE           = TDSVARYCHAR;
        DB_CLIENT_TYPE         = TDSCHAR;
        CALL DESCRIBE_COLUMN;

        DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_ED);
        DB_COLUMN_NAME_HV_PTR  = ADDR(CN_ED);
        WRKLEN1                = STG(EMPLOYEE_ED);
        WRKLEN2                = STG(CN_ED);
        DB_HOST_TYPE           = TDSINT2;
        DB_CLIENT_TYPE         = TDSINT2;
        CALL DESCRIBE_COLUMN;
/*      ------------------------------------------------------------*/
/*      Get the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
        CALL TDINFUDT (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       GWL_INFUDT_USER_TYPE);
/*      ------------------------------------------------------------*/
/*      Set the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
        CALL TDSETUDT (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       GWL_INFUDT_USER_TYPE);
/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.    */
/*      ------------------------------------------------------------*/
        DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_JC);
        DB_COLUMN_NAME_HV_PTR  = ADDR(CN_JC);
        WRKLEN1                = STG(EMPLOYEE_JC);
        WRKLEN2                = STG(CN_JC);
        DB_HOST_TYPE           = TDSDECIMAL;
        DB_CLIENT_TYPE         = TDSFLT8;
        CALL DESCRIBE_COLUMN;
/*      ------------------------------------------------------------*/
/*      We must inform the Server Library how many decimal places   */
/*      are in the EMPLOYEE_JC column.                              */
/*      ------------------------------------------------------------*/
        CALL TDSETBCD (GWL_PROC, GWL_RC,
                       TDS_OBJECT_COL,
                       CTR_COLUMN,
                       TDS_DEFAULT_LENGTH,
                       GWL_SETBCD_SCALE);
```

```
/*      ------------------------------------------------------------*/
/*      Demonstrate getting decimal column information.             */
/*      ------------------------------------------------------------*/
      CALL TDINFBCD (GWL_PROC, GWL_RC,
                     TDS_OBJECT_COL,
                     CTR_COLUMN,
                     GWL_INFBCD_LENGTH,
                     GWL_INFBCD_SCALE);
/*      ------------------------------------------------------------*/
/*      Here we intend to use TDCONVRT to convert from TDSDECIMAL to*/
/*      TDSMONEY, so we point TDESCRIB to the output of TDCONVRT,   */
/*      rather than the original input.                             */
/*      ------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR    = ADDR(WRK_EMPLOYEE_SAL);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_SAL);
      WRKLEN1               = STG(WRK_EMPLOYEE_SAL);
      WRKLEN2               = STG(CN_SAL);
      DB_HOST_TYPE          = TDSMONEY;
      DB_CLIENT_TYPE        = TDSMONEY;
      CALL DESCRIBE_COLUMN;
/*------------------------------------------------------------------*/
SEND_ROWS:
/*------------------------------------------------------------------*/
      DO WHILE(^ ALL_DONE);
         CALL FETCH_AND_SEND_ROWS;
      END;
/*------------------------------------------------------------------*/
END_OF_QUERY:
/*------------------------------------------------------------------*/
/*      ------------------------------------------------------------*/
/*      close cursor                                                */
/*      ------------------------------------------------------------*/
      EXEC SQL CLOSE ECURSOR;
/*      ------------------------------------------------------------*/
/*      update return parameter with nr of rows fetched             */
/*      ------------------------------------------------------------*/
      CALL TDSETPRM (GWL_PROC, GWL_RC,
                     GWL_SETPRM_ID,
                     GWL_SETPRM_TYPE,
                     GWL_SETPRM_DATA_L,

                     PARM_RETURN_ROWS,
                     GWL_SETPRM_USER_DATA);

      GO TO END_PROGRAM;
/*------------------------------------------------------------------*/
```

```
FETCH_AND_SEND_ROWS: PROC;
/*-------------------------------------------------------------------*/
       EXEC SQL FETCH ECURSOR INTO :EMPLOYEE_FIELDS;

       IF SQLCODE = 0 THEN
       DO;
/*           ---------------------------------------------------------*/
/*           Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.    */
/*           ---------------------------------------------------------*/
           WRKLEN1 = STG(EMPLOYEE_SAL);
           WRKLEN2 = STG(WRK_EMPLOYEE_SAL);
CALL TDCONVRT (GWL_PROC, GWL_RC,
                         GWL_CONVRT_SCALE,
                         TDSDECIMAL,
                         WRKLEN1,
                         EMPLOYEE_SAL,
                         TDSMONEY,
                         WRKLEN2,
                         WRK_EMPLOYEE_SAL);
/*           ---------------------------------------------------------*/
/*           Do not send trailing blanks of EMPLOYEE_LNM              */
/*           ---------------------------------------------------------*/
           WRKLEN1 = LENGTH(EMPLOYEE_LNM);
           CTR_COLUMN = 2;

           WRK_BLANKS_SS = 1;
LOOP:       DO WHILE(WRK_BLANKS_SS <= WRKLEN1);
             IF SUBSTR(EMPLOYEE_LNM, WRK_BLANKS_SS, 1) = ' ' THEN DO;
               LEAVE LOOP;
             END;

             WRK_BLANKS_SS = WRK_BLANKS_SS + 1;
           END LOOP;

           IF (WRK_BLANKS_SS <= WRKLEN1) THEN DO;
             CALL TDSETLEN (GWL_PROC, GWL_RC,
                            CTR_COLUMN,
                            WRK_BLANKS_SS - 1);
           END;
/*           ---------------------------------------------------------*/
/*           send a row to the client                                 */
/*           ---------------------------------------------------------*/
           CALL TDSNDROW (GWL_PROC, GWL_RC);
           PARM_RETURN_ROWS = PARM_RETURN_ROWS + 1;

           IF GWL_RC = TDS_CANCEL_RECEIVED THEN
```

```
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;
        END;

        ELSE IF SQLCODE = +100 THEN
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;

        ELSE IF SQLCODE < 0 THEN
        DO;
            ALL_DONE = ALL_DONE_YES;
            CALL FETCH_ERROR;
        END;

        RETURN;

END FETCH_AND_SEND_ROWS;
```

Usage

TDCONVRT

A server application uses this function to convert from a mainframe datatype to a datatype that can be used by a DB-Library or Client-Library client. See "Datatypes" on page 36 for more information about particular datatypes and datatype conversions. For details about DB-Library datatypes, see the Open Client DB-Library *Reference Manual*. For details about Client-Library datatypes, see the Open Client Client-Library *Reference Manual*.

Note Most Gateway-Library-to-Client-Library datatype conversions can be done more efficiently with TDESCRIB and TDSETPRM, which perform automatic data conversions. For more information, see TDESCRIB on page 89 and TDSETPRM on page 194.

- This function converts a single variable each time it executes.
- If several columns in a single result row will be converted, an application must issue a separate TDCONVRT call for each column that will be converted before it sends the row to a client. If several rows of data are sent to the client, the application must issue a separate TDCONVRT call for every column that needs conversion in each row, before it issues a TDSNDROW call for that row.

- If TDESCRIB follows TDCONVRT, be sure that the TDESCRIB *HOST_VARIABLE_NAME* argument corresponds to the TDCONVRT *RESULT_VARIABLE* rather than the *SOURCE_VARIABLE*.

Datatype conversions

Table 3-4 on page 82 lists the conversions you can perform with TDCONVRT.

Table 3-4: Datatype conversions performed by TDCONVRT

Source datatype	Result datatype	Notes
TDSCHAR	TDSVARYCHAR	Performs EBCDIC and ASCII conversion. Pads TDSCHAR fields with blanks. When converting TDSCHAR to Sybase numeric and decimal, specify 35 as destination length. *OUTLEN* shows the actual length.
TDSCHAR	TDSLONGVARCHAR	
TDSCHAR	TDSMONEY	
TDSCHAR	TDSNUMERIC	
TDSCHAR	TDS_SYBASE_DECIMAL	
TDSCHAR	TDS_PACKED_DECIMAL	
TDSVARYCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSMONEY	
TDSLONGVARCHAR	TDSCHAR	
TDSLONGVARCHAR	TDSTEXT	
TDSLONGVARCHAR	TDSVARYCHAR	
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	
TDSFLT4	TDSFLT8	Pads with zeroes.
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	
TDSGRAPHIC	TDSCHAR	Used with Japanese double-byte character sets. Pads TDSCHAR fields with blanks.
TDSGRAPHIC	TDSVARYCHAR	
TDSVARYGRAPHIC	TDSCHAR	
TDSVARYGRAPHIC	TDSVARYCHAR	
TDSLONGVARBIN	TDSIMAGE	

Source datatype	Result datatype	Notes
TDSNUMERIC TDSNUMERIC	TDSCHAR TDS_PACKED_DECIMAL	When converting Sybase numeric and decimal to char, specify destination length as precision + 2, or precision +3 if precision=scale for leading zero. When converting from numeric to TDS_PACKED_DECIMAL the destination should supply the same precision and scale as the source. For numeric (15,5) specify destination as FIXED DEC (15,5).
TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL	TDSCHAR TDSVARYCHAR TDSMONEY TDSNUMERIC TDSFLT4 TDSFLT8 TDS_SYBASE_DECIMAL	When converting packed decimal to character values, change the length to allow for unpacking, leading or trailing zeros, the sign and the decimal point. When converting TDS_PACKED_DECIMAL to Sybase numeric and decimal, specify 35 as the destination length. *OUTLEN* shows the actual length of the numeric field.
TDS_SYBASE_DECIMAL TDS_SYBASE_DECIMAL	TDSCHAR TDS_PACKED_DECIMAL	When converting Sybase numeric and decimal to char, specify destination length as precision + 2, or precision +3 if precision=scale for leading zero.

Warning! The results of decimal-to-character type conversions are no longer formatted in SQL Processor Using File Input (SPUFI) style. See “Converting packed decimal to character data” on page 42 for an explanation of how the results now handle leading and trailing zeroes.

DB2 users

For VARCHAR strings:

- Treat VARCHAR strings as TDSVARYCHAR. You can safely convert these strings to DB-library VARYCHAR or CHAR.
- For 255-byte LONG VARCHAR strings:

    - Treat these strings as TDSVARYCHAR. TDSVARYCHAR strings can be up to 255 bytes in length. You can safely convert these strings to DB-library VARYCHAR or CHAR.
- For longer LONG VARCHAR strings (256 or more bytes):
    - Treat these strings as TDSLONGVARCHAR. When converting long varchar data to a client datatype, you have three options:
        1. If the client program supports TEXT datatypes, you can convert the string to TDSTEXT before sending it to the client. TDSTEXT is a variable-length datatype containing up to 2,147,483,647 bytes.
        2. If the client is an Open Client 10.0 program, you can send the data as TDSLONGVARCHAR. The Client-Library datatype CS_LONGCHAR has a maximum length of 2,147,483,647 bytes.
        3. If the truncation option is set during customization, you can send the string as TDSVARYCHAR. If you choose this option, the data is truncated. However, if the truncation option is not set, and you try to convert these strings to TDSVARYCHAR, an error is returned.

For binary strings:

- If the client program supports IMAGE datatypes, you can convert the string to TDSIMAGE before sending it to the client. TDSIMAGE is a variable-length datatype containing up to 2,147,483,647 bytes.
- If the client is a Client-Library 10.0 program, you can send the data as TDSLONGVARBIN. The Client-Library datatype CS_LONGBINARY has a maximum length of 2,147,483,647 bytes.

See also

Related functions

- TDESCRIB on page 89
- TDRCVPRM on page 162
- TDSETBCD on page 181
- TDSETPRM on page 194

Related topics

- “Datatypes” on page 36

TDCURPRO

Description

Retrieves or sets information about a cursor.

Syntax

```
%INCLUDE SYGWPLI;

 01 TDPROC          FIXED BIN (31),
 01 RETCODE         FIXED BIN (31),
 01 ACTION      FIXED BIN (31),
 01 CURSOR_DESC from SYGWPLI;

CALL TDCURPRO (TDPROC, RETCODE, ACTION,
          CURSOR_DESC);
```

The *CURSOR_DESC* structure is defined in SYGWPLI as follows:

```
CURSOR_ID              FIXED BIN (31),
NUMBER_OF_UPDATE_COLUMNS FIXED BIN (31),
FETCH_COUNT            FIXED BIN (31),
CURSOR_STATUS          FIXED BIN (31),
CURSOR_COMMAND         FIXED BIN (31),
COMMAND_OPTIONS        FIXED BIN (31),
FETCH_TYPE             FIXED BIN (31),
ROW_OFFSET             FIXED BIN (31),
CURSOR_NAME_LENGTH     FIXED BIN (31),
CURSOR_NAME            CHAR (n),
TABLE_NAME_LENGTH      FIXED BIN (31),
TABLE_NAME             CHAR(n);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-5 on page 86.

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:

TDS_GET (33)	Retrieves cursor information.
TDS_SET (34)	Specifies cursor information.

CURSOR_DESC
(I/O) A *CURSOR_DESC* structure containing information in the following fields:

This field	Contains this information
CURSOR_ID	The cursor identifier.
NUMBER_OF_UPDATE_ COLUMNS	The number of columns in a cursor update clause.
FETCH_COUNT	The current row fetch count for this cursor; that is, the number of rows that are sent to the client in response to a TDS_CURSOR_FETCH command.
CURSOR_STATUS	The status of the current cursor.
CURSOR_COMMAND	The current cursor command type.
COMMAND_OPTIONS	Any options associated with the cursor command.
FETCH_TYPE	The type of fetch requested by a client.
ROW_OFFSET	The row position for TDS_ABSOLUTE or TDS_RELATIVE fetches.
CURSOR_NAME_LENGTH	The length of the cursor name in CURSOR_NAME.
CURSOR_NAME	The name of the current cursor.
TABLE_NAME_LENGTH	The length of the tablename in TABLE_NAME.
TABLE_NAME	The table name associated with a cursor update or delete command.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-5.

Table 3-5: TDCURPRO return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_BUFFLEN_GREATER_TYPE (-191)	TDYNAMIC: The size of the buffer is greater than the dynamic SQL-type field being retrieved.
TDS_BUFFLEN_LESS_TYPE (-192)	TDYNAMIC: The size of the buffer is too small to return a dynamic SQL-type field.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CMD_NOT_GET_SET (-190)	The value of the *ACTION* argument is invalid. It should be either TDS_GET or TDS_SET.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.

Return value	Meaning
TDS_CURSOR_ALREADY_OPEN (-74)	Cursor already open. You cannot open the same cursor more than once.
TDS_CURSOR_NOT_CLOSED (-73)	Cursor is still active (deallocate without close first).
TDS_CURSOR_NOT_DECLARED (-70)	A cursor must be declared before it can be opened.
TDS_CURSOR_NOT_OPEN (-72)	Cursor not open. A cursor must be open before a fetch, close, delete, or update.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_CURCLOSOPTION (-182)	A “closed” cursor command specified an invalid option. The Gateway-Library transaction received a “closed” cursor command, but the value of the OPTION field of the *CURSOR_DESC* structure is invalid. Valid options are TDS_CUR_UNUSED and TDS_CUR_DEALLOC.
TDS_INVALID_CURDECLOPTION (-183)	A declare cursor command has an invalid option specified. The Gateway-Library transaction received a declare cursor command, but the value of the OPTION field of the *CURSOR_DESC* structure is invalid. Valid options are TDS_CUR_UNUSED and TDS_CUR_DEALLOC.
TDS_INVALID_CURDECLSTAT (-184)	Illegal cursor declare option.
TDS_INVALID_CURINFCMD (-195)	Illegal cursor information command.
TDS_INVALID_CURINFSTAT (-185)	Illegal cursor information status.
TDS_INVALID_CUROPENSTAT (-187)	Illegal cursor open status.
TDS_INVALID_CURSOR_COMMAND (-194)	The cursor command is not declare, open, fetch, delete, update, or close.
TDS_INVALID_CURSOR_FSM (-78)	Invalid cursor state.
TDS_INVALID_CURUPDSTAT (-186)	Illegal cursor update status.
TDS_INVALID_OP_TYPE (-193)	Invalid dynamic SQL operation.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_NO_CURRENT_CURSOR (-200)	No cursor is associated with the current transaction.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Usage

TDCURPRO

An Open ServerConnect application uses TDCURPRO to exchange active cursor information with a client.

- A transaction first calls TDCURPRO, with an *ACTION* of TDS_GET to retrieve the client cursor command and other information about the cursor (for example, the requested fetch count). The *CURSOR_DESC* structure provides this information.
- After processing the client command, the transaction calls TDCURPRO with an *ACTION* of TDS_SET to return acknowledgment and/or updated cursor information to the client.
- Each type of cursor command requires a distinct response from the Open ServerConnect application. Cursor commands and the responses are discussed in "Types of cursor commands" on page 22.

An application can also read in parameters or send back result rows, depending on the circumstances.

- An application can call TDCURPRO for any cursor by specifying the cursor in the CURSOR_ID field of the *CURSOR_DESC* structure. Cursors need not be called in any particular order.
- The CURSOR_COMMAND field in the *CURSOR_DESC* structure indicates the command to be processed.
- When a client declares a new cursor (CURSOR_COMMAND is TDS_CURSOR_DECLARE), the client provides a cursor name, but not a cursor ID. It is the responsibility of the Open ServerConnect application to assign a unique cursor ID to the new cursor and return that ID to the client.

  To do this:

  a Specify TDS_SET for the *ACTION* argument.

  b Specify the new cursor ID in the CURSOR_ID field in the *CURSOR_DESC* structure.

  c Return this information to the client.

  Both the client and the Open ServerConnect applications must subsequently refer to this cursor by its ID rather than by its name.
- The application must acknowledge all cursor commands except fetch, update, and delete by sending back a cursor information command.

To do this specify TDS_SET in the *ACTION* argument. This is the very first piece of information the application sends back after receiving a cursor command. The application sets the cursor ID. This information comes back on every command.

For example, after receiving a close cursor request Open ServerConnect sets CURSOR_COMMAND to TDS_CURSOR_INFO and CURSOR_STATUS to TDS_CURSTAT_CLOSED.

Multiple cursor commands per transaction invocation are not allowed. To process multiple commands, use the long-running transaction, accepting each new command request with TDGETREQ.

See also

Related functions

- TDACCEPT on page 68
- TDGETREQ on page 103

TDESCRIB

Description

Describes a column in a result row and the mainframe server program variable where it is stored.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC                PTR,
01 RETCODE               FIXED BIN(31),
01 COLUMN_NUMBER         FIXED BIN(31),
01 HOST_VARIABLE_TYPE    FIXED BIN(31),
01 HOST_VARIABLE_MAXLEN  FIXED BIN(31),
01 HOST_VARIABLE_NAME    CHAR(n),
01 NULL_INDICATOR_VARIABLE  FIXED BIN(15) INIT(0),
01 NULLS_ALLOWED         FIXED BIN(31),
01 COLUMN_TYPE           FIXED BIN(31),
01 COLUMN_MAXLEN         FIXED BIN(31),
01 COLUMN_NAME           CHAR(n),
01 COLUMN_NAME_LENGTH    FIXED BIN(31);

CALL TDESCRIB (TDPROC, RETCODE, COLUMN_NUMBER,
HOST_VARIABLE_TYPE,
          HOST_VARIABLE_MAXLEN,
          HOST_VARIABLE_NAME,
          NULL_INDICATOR_VARIABLE, NULLS_ALLOWED,
          COLUMN_TYPE, COLUMN_MAXLEN,
           COLUMN_NAME, COLUMN_NAME_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-6 on page 91.

COLUMN_NUMBER

(I) Number of the column that is being described. Columns are numbered sequentially. The first column in a row is number 1.

HOST_VARIABLE_TYPE

(I) Datatype of *HOST_VARIABLE_NAME*, the host program variable where the data for this column is stored. If you use TDCONVRT to convert from one datatype to another, this is the *RESULT_TYPE*.

HOST_VARIABLE_MAXLEN

(I) Maximum length of the host program variable. This is the value of (n) in the definition statement for *HOST_VARIABLE_NAME*.

For TDSVARYCHAR, TDSVARYBIN, and TDSVARYGRAPHIC variables, this length does not include the 2 bytes for the “LL” length specification. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the actual length.

HOST_VARIABLE_NAME

(I) Host program variable that contains the data for this column.

You must name a different variable for each column to be described.

If you use TDCONVRT to convert from one datatype to another, this is the *RESULT_VARIABLE*. If the datatype is TDSVARYCHAR, TDSVARYBIN, or TDSVARYGRAPHIC, this is the name of a structure that includes the “LL” length specification.

NULL_INDICATOR_VARIABLE

(I) Host program variable that contains the NULL indicator for this column. When the value in this variable is negative, TDSNDROW sends a NULL value for this column. Note that this variable is a halfword.

If *NULLS_ALLOWED* is TDS_FALSE, this argument is ignored.

NULLS_ALLOWED

(I) Null permission indicator. Indicates whether NULLs are allowed for this column. Assign this argument one of the following values:

TDS_TRUE (1)	NULLs are allowed.
TDS_FALSE (0)	NULLs are not allowed.

Note NULLs are typically used with DB2.

COLUMN_TYPE

(I) Open Client datatype of the column. This is the datatype used by the client application.

COLUMN_MAXLEN

(I) Maximum length of the column data. For variable-length datatypes, this argument represents the maximum length for a value of that datatype. For fixed-length datatypes (TDSINT*n*, TDSFLT*n*), this argument is ignored.

See "Comments" in this section for cautions about converting packed decimal datatypes and Japanese double-byte characters.

COLUMN_NAME

(I) Name of the column with the data that is being returned.

COLUMN_NAME_LENGTH

(I) Actual length of the column name.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-6.

Table 3-6: TDESCRIB return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_DUPLICATE_ENTRY (-9)	Duplicate column description. You attempted to describe the same column twice with a TDESCRIB statement. The operation failed.

Return value	Meaning
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).
TDS_INVALID_DATA_CONVERSION (-172)	Incompatible datatypes. The source datatype cannot be converted into the requested result datatype.
TDS_INVALID_DATA_TYPE (-171)	Illegal datatype. A sybase datatype supplied in the call is not supported and the conversion can not be completed.
TDS_INVALID_ID_VALUE (-10)	The specified column or parameter number is greater than the system maximum. Sybase allows as many columns per table result and parameters per RPC as the system maximum.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *COLUMN_MAXLEN* argument is too short.
TDS_INVALID_NAMELENGTH (-179)	Invalid name length. The length specified for the column, parameter, message, or server name is invalid.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates a typical use of TDESCRIB to describe a column in a return row. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*-------------------------------------------------------------------*/
SETUP_REPLY_COLUMNS:
/*-------------------------------------------------------------------*/
       DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_FNM);
       DB_COLUMN_NAME_HV_PTR  = ADDR(CN_FNM);
       WRKLEN1                = STG(EMPLOYEE_FNM)-2;
       WRKLEN2                = STG(CN_FNM);
```

```
      DB_HOST_TYPE          = TDSVARYCHAR;
      DB_CLIENT_TYPE        = TDSVARYCHAR;
      CALL DESCRIBE_COLUMN;

/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR) */
/*      to DBCHAR.                                                  */
/*      ------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_LNM);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_LNM);
      WRKLEN1               = STG(EMPLOYEE_LNM)-2;
      WRKLEN2               = STG(CN_LNM);
      DB_HOST_TYPE          = TDSVARYCHAR;
      DB_CLIENT_TYPE        = TDSCHAR;
      CALL DESCRIBE_COLUMN;

      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_ED);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_ED);
      WRKLEN1               = STG(EMPLOYEE_ED);
      WRKLEN2               = STG(CN_ED);
      DB_HOST_TYPE          = TDSINT2;
      DB_CLIENT_TYPE        = TDSINT2;
      CALL DESCRIBE_COLUMN;
/*      ------------------------------------------------------------*/
/*      Get the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
      CALL TDINFUDT (GWL_PROC, GWL_RC,
                     CTR_COLUMN,
                     GWL_INFUDT_USER_TYPE);
/*      ------------------------------------------------------------*/
/*      Set the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
      CALL TDSETUDT (GWL_PROC, GWL_RC,
                     CTR_COLUMN,
                     GWL_INFUDT_USER_TYPE);
/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.    */
/*      ------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_JC);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_JC);
      WRKLEN1               = STG(EMPLOYEE_JC);
      WRKLEN2               = STG(CN_JC);
      DB_HOST_TYPE          = TDSDECIMAL;
      DB_CLIENT_TYPE        = TDSFLT8;
      CALL DESCRIBE_COLUMN;
```

```
/*      ------------------------------------------------------------*/
/*      We must inform the Server Library how many decimal places   */
/*      are in the EMPLOYEE_JC column.                              */
/*      ------------------------------------------------------------*/
      CALL TDSETBCD (GWL_PROC, GWL_RC,
                     TDS_OBJECT_COL,
                     CTR_COLUMN,
                     TDS_DEFAULT_LENGTH,
                     GWL_SETBCD_SCALE);
/*      ------------------------------------------------------------*/
/*      Demonstrate getting decimal column information.             */
/*      ------------------------------------------------------------*/
      CALL TDINFBCD (GWL_PROC, GWL_RC,
                     TDS_OBJECT_COL,
                     CTR_COLUMN,
                     GWL_INFBCD_LENGTH,
                     GWL_INFBCD_SCALE);
/*      ------------------------------------------------------------*/
/*      Here we intend to use TDCONVRT to convert from TDSDECIMAL to*/
/*      TDSMONEY, so we point TDESCRIB to the output of TDCONVRT,   */
/*      rather than the original input.                             */
/*      ------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR     = ADDR(WRK_EMPLOYEE_SAL);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_SAL);
      WRKLEN1                = STG(WRK_EMPLOYEE_SAL);
      WRKLEN2                = STG(CN_SAL);
      DB_HOST_TYPE           = TDSMONEY;
      DB_CLIENT_TYPE         = TDSMONEY;
      CALL DESCRIBE_COLUMN;
/*--------------------------------------------------------------------*/
SEND_ROWS:
/*--------------------------------------------------------------------*/
      DO WHILE(^ ALL_DONE);
         CALL FETCH_AND_SEND_ROWS;
      END;
/*--------------------------------------------------------------------*/
      END_OF_QUERY:
/*--------------------------------------------------------------------*/
/*      ------------------------------------------------------------*/
/*      close cursor                                                */
/*      ------------------------------------------------------------*/
      EXEC SQL CLOSE ECURSOR;
/*      ------------------------------------------------------------*/
/*      update return parameter with nr of rows fetched             */
/*      ------------------------------------------------------------*/
      CALL TDSETPRM (GWL_PROC, GWL_RC,
```

```
                        GWL_SETPRM_ID,
                        GWL_SETPRM_TYPE,
                        GWL_SETPRM_DATA_L,
                        PARM_RETURN_ROWS,
                        GWL_SETPRM_USER_DATA);

      GO TO END_PROGRAM;
/*-------------------------------------------------------------------*/
FETCH_AND_SEND_ROWS: PROC;
/*-------------------------------------------------------------------*/
      EXEC SQL FETCH ECURSOR INTO :EMPLOYEE_FIELDS;

      IF SQLCODE = 0 THEN
      DO;
/*          ----------------------------------------------------------*/
/*          Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.    */
/*          ----------------------------------------------------------*/
          WRKLEN1 = STG(EMPLOYEE_SAL);
          WRKLEN2 = STG(WRK_EMPLOYEE_SAL);

          CALL TDCONVRT (GWL_PROC, GWL_RC,
                         GWL_CONVRT_SCALE,
                         TDSDECIMAL,
                         WRKLEN1,
                         EMPLOYEE_SAL,
                         TDSMONEY,
                         WRKLEN2,
                         WRK_EMPLOYEE_SAL);
/*          ----------------------------------------------------------*/
/*          Do not send trailing blanks of EMPLOYEE_LNM              */
/*          ----------------------------------------------------------*/
          WRKLEN1 = LENGTH(EMPLOYEE_LNM);
          CTR_COLUMN = 2;

          WRK_BLANKS_SS = 1;
LOOP:       DO WHILE(WRK_BLANKS_SS <= WRKLEN1);
            IF SUBSTR(EMPLOYEE_LNM, WRK_BLANKS_SS, 1) = ' ' THEN DO;
              LEAVE LOOP;
            END;

            WRK_BLANKS_SS = WRK_BLANKS_SS + 1;
          END LOOP;

          IF (WRK_BLANKS_SS <= WRKLEN1) THEN DO;
            CALL TDSETLEN (GWL_PROC, GWL_RC,
                           CTR_COLUMN,
```

```
                              WRK_BLANKS_SS - 1);
            END;
/*-------------------------------------------------------------------*/
/*          send a row to the client                                 */
/*-------------------------------------------------------------------*/
            CALL TDSNDROW (GWL_PROC, GWL_RC);
            PARM_RETURN_ROWS = PARM_RETURN_ROWS + 1;
            IF GWL_RC = TDS_CANCEL_RECEIVED THEN
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;
       END;
       ELSE IF SQLCODE = +100 THEN
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;
       ELSE IF SQLCODE < 0 THEN
       DO;
            ALL_DONE = ALL_DONE_YES;
            CALL FETCH_ERROR;
       END;
       RETURN;
END FETCH_AND_SEND_ROWS;
/*-------------------------------------------------------------------*/
GET_PARM_INFO: PROC;
/*-------------------------------------------------------------------*/
       CALL TDINFPRM (GWL_PROC, GWL_RC,
                      GWL_INFPRM_ID,
                      GWL_INFPRM_TYPE,
                      GWL_INFPRM_DATA_L,
                      GWL_INFPRM_MAX_DATA_L,
                      GWL_INFPRM_STATUS,
                      GWL_INFPRM_NAME,
                      GWL_INFPRM_NAME_L,
                      GWL_INFPRM_USER_DATA);
       RETURN;
END GET_PARM_INFO;
/*-------------------------------------------------------------------*/
DESCRIBE_COLUMN: PROC;
/*-------------------------------------------------------------------*/
       CTR_COLUMN = CTR_COLUMN +1;
       CALL TDESCRIB (GWL_PROC, GWL_RC,
                      CTR_COLUMN,
                      DB_HOST_TYPE,
                      WRKLEN1,
                      DB_DESCRIBE_HV,
```

```
                                DB_NULL_INDICATOR,
                                TDS_FALSE,
                                DB_CLIENT_TYPE,
                                WRKLEN1,
                                DB_COLUMN_NAME_HV,
                                WRKLEN2);
    RETURN;
    END DESCRIBE_COLUMN;
```

Usage

TDESCRIB

A server application uses this function to describe a column that is returned to the client and the host program variable where the column data is stored.
A server application uses this function to describe a column that is returned to the client and the host program variable where the column data is stored.

A server application uses this function to describe a column that is returned to the client and the host program variable where the column data is stored.

Datatype conversions

Table 3-7 on page 97 shows which conversions are performed automatically when TDESCRIB is called.

Table 3-7: Datatype conversions performed by TDESCRIB

Source datatype	Result datatype	Notes
TDSCHAR	TDSVARYCHAR	Performs EBCDIC and ASCII conversion.
TDSCHAR	TDSLONGVARCHAR	
TDSCHAR	TDSMONEY	Pads TDSCHAR fields with blanks.
TDSCHAR	TDSNUMERIC	
TDSCHAR	TDS_SYBASE_DECIMAL	When converting TDSCHAR to Sybase numeric and decimal, specify 35 as destination length. *OUTLEN* shows the actual length.
TDSCHAR	TDS_PACKED_DECIMAL	
TDSVARYCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSMONEY	
TDSLONGVARCHAR	TDSCHAR	
TDSLONGVARCHAR	TDSTEXT	
TDSLONGVARCHAR	TDSVARYCHAR	
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	
TDSFLT4	TDSFLT8	Pads with zeroes.
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	

Source datatype	Result datatype	Notes
TDSGRAPHIC TDSGRAPHIC TDSVARYGRAPHIC TDSVARYGRAPHIC	TDSCHAR TDSVARYCHAR TDSCHAR TDSVARYCHAR	Used with Japanese double-byte character sets. Pads TDSCHAR fields with blanks.
TDSLONGVARBIN	TDSIMAGE	
TDSNUMERIC TDSNUMERIC	TDSCHAR TDS_PACKED_DECIMAL	When converting Sybase numeric and decimal to char, specify destination length as precision + 2, or precision +3 when precision=scale. When converting from numeric to TDS_PACKED_DECIMAL the destination should supply the same precision and scale as the source. For numeric (15,5) specify destination as FIXED DEC (15,5).
TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL TDS_PACKED_DECIMAL	TDSCHAR TDSVARYCHAR TDSMONEY TDSNUMERIC TDSFLT4 TDSFLT8 TDS_SYBASE_DECIMAL	When converting packed decimal to character values, change the length to allow for unpacking, leading or trailing zeros, the sign and the decimal point. When converting TDS_PACKED_DECIMAL to Sybase numeric and decimal, specify 35 as the destination length. *OUTLEN* shows the actual length of the numeric field.
TDS_SYBASE_DECIMAL TDS_SYBASE_DECIMAL	TDSCHAR TDS_PACKED_DECIMAL	When converting Sybase numeric and decimal to char, specify destination length as precision + 2, or precision +3 when precision=scale (for leading zero).

- When converting packed decimal data, the *COLUMN_MAXLEN* must allow for:
  - Unpacking
  - Leading and trailing zeros
  - Sign and decimal point

  A suggested formula for unpacking is:

```
Result Length =  (2 * Source Length) - 1.
```

- Always use TDSETBCD when describing Sybase decimal and numeric columns. Assign the following:
  - Precision to *BCD_LENGTH*
  - Scale to *BCD_NUMBER_DECIMAL_PLACES*
- See "Datatypes" on page 36 for more information about datatypes supported by Gateway-Library.

For Japanese users

The Japanese Conversion Module (JCM) automatically converts column names from the character set used at the mainframe server to that specified by the client in the login packet.

- When converting Japanese characters, TDESCRIB changes the length of the column name to the length required by the client character set, which can be different from the length of the column name at the mainframe.

  For example, the length of a hankaku katakana character is 1 in the mainframe character set, IBM-Kanji, and 2 in the workstation character set, eucjis.

  To learn more, see the discussion of length considerations in "Processing Japanese client requests" on page 56.

See also

Related functions

- TDCONVRT on page 75
- TDSNDROW on page 227

Related topics

- "Datatypes" on page 36
- "Processing Japanese client requests" on page 56

TDFREE

Description

Frees up a previously allocated *TDPROC* structure after returning results to a client.

Syntax

```
%INCLUDE SYGWPLI;
```

```
01 TDPROC  PTR,
01 RETCODE FIXED BIN(31);

CALL TDFREE  (TDPROC, RETCODE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-8.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-8 on page 100.

Table 3-8: TDFREE return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

Example 1

The following code fragment illustrates the use of TDINIT, TDACCEPT, TDSNDDON, and TDFREE at the beginning and end of a Gateway-Library program. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*-------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-------------------------------------------------------------*/
/*      -------------------------------------------------------*/
/*      reset db2 error handlers                               */
/*      -------------------------------------------------------*/
       EXEC SQL WHENEVER SQLWARNING CONTINUE;
       EXEC SQL WHENEVER SQLERROR   CONTINUE;
       EXEC SQL WHENEVER NOT FOUND  CONTINUE;
/*      -------------------------------------------------------*/
/*      establish gateway environment                          */
/*      -------------------------------------------------------*/
       CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);
/*      -------------------------------------------------------*/
/*      accept client request                                  */
/*      -------------------------------------------------------*/
```

```
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);
   /*-------------------------------------------------------------------*/
   /*  (BODY OF PROGRAM)                                                */
   /*-------------------------------------------------------------------*/
        GO TO END_PROGRAM;
   /*-------------------------------------------------------------------*/
   END_PROGRAM:
   /*-------------------------------------------------------------------*/
        IF SEND_DONE = SEND_DONE_OK THEN
            WRK_DONE_STATUS = TDS_DONE_COUNT;

        ELSE
        DO;
            WRK_DONE_STATUS  = TDS_DONE_ERROR;
            PARM_RETURN_ROWS = 0;
        END;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       WRK_DONE_STATUS,
                       PARM_RETURN_ROWS,
                       TDS_ZERO,
                       TDS_ENDRPC);

        CALL TDFREE (GWL_PROC, GWL_RC);
        EXEC CICS RETURN;

 END SYSAMP1;
```

Example 2

The following code fragment illustrates the use of TDFREE and TDTERM at the end of a Gateway-Library program. This example is taken from the sample program SYIXSAM1, listed in Appendix D, "Sample RPC Application for IMS TM (Explicit)."

```
FREE_STORAGE: PROC;

        CALL TDFREE (GWL_PROC, GWL_RC);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDFREE  ';
              CALL_MSG = 'ERROR IN CALL RC=';
              CALL_RC  = GWL_RC;
              PUT FILE(SYSPRINT) DATA(ERROR_MSG);
           END;
```

```
            CALL TDTERM (GWL_INIT_HANDLE, GWL_RC);

            IF GWL_RC ^= 0 THEN
               DO;
                  CALL_NAME = 'TDTERM  ';
                  CALL_MSG = 'ERROR IN CALL RC=';
                  CALL_RC  = GWL_RC;
                  PUT FILE(SYSPRINT) DATA(ERROR_MSG);
               END;

   END FREE_STORAGE;
```

Usage

TDFREE

An application calls this function to clean up and deallocate the *TDPROC* structure defined for this connection in TDACCEPT (For TCP/IP applications, this closes the socket).

- TDFREE does not free up the *IHANDLE*.

  Under CICS: the *IHANDLE* is automatically freed when the transaction ends.

  Under IMS TM and MVS: the transaction must call TDTERM to free the *IHANDLE*.

- Typically, a transaction calls TDFREE either at the end of a transaction or after TDRESULT returns TDS_CONNECTION_TERMINATED or TDS_CONNECTION_FAILED.

- The last call in an IMS TM program after it processes all requests must be TDTERM. It frees all remaining resources, including the *IHANDLE*, in preparation for program termination. We strongly recommend ending all programs with a TDTERM call.

See also

Related functions

- TDACCEPT on page 68
- TDGETREQ on page 103
- TDINIT on page 148
- TDRESULT on page 174

TDGETREQ

Description

Accepts the next request in a long-running transaction.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC       PTR,
01 RETCODE      FIXED BIN(31),
01 WAIT_OPTION  FIXED BIN(31),
01 REQUEST_TYPE FIXED BIN(31),
01 TRAN_NAME    CHAR(n);

CALL TDGETREQ (TDPROC, RETCODE, WAIT_OPTION,
               REQUEST_TYPE, TRAN_NAME);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-9 on page 104.

WAIT_OPTION

(I) Wait/do not wait indicator. Indicates what the application should do after a TDGETREQ if no request is present:

(1) wait for a new request to arrive, or

(2) terminate immediately.

Assign this argument one of the following values:

TDS_TRUE (1)	Wait for input.
TDS_FALSE (0)	Do not wait for input.

Under CICS and MVS: We recommend always coding TDS_FALSE. Coding TDS_FALSE ends the transaction and frees resources if there is nothing left to do. Coding TDS_TRUE causes the transaction to wait.

Under IMS TM: The *WAIT_OPTION* tells the transaction what to do when the message queue is empty: wait for another request to appear on the queue, or end the transaction.

Note To use TDGETREQ properly under the IMS TM implicit API, the transaction must be a WPI transaction, or the message region that the transaction runs in must have PWFI=Y (Pseudo-Wait-For-Input) specified.

REQUEST_TYPE

(O) Type of request to be accepted. Returns one of the following values:

TDS_LANGUAGE_EVENT (1)	Current request is a language request.
TDS_RPC_EVENT (3)	Current request is an RPC.
TDS_DYNAMIC_EVENT (4)	Current request is a dynamic SQL request.
TDS_CURSOR_EVENT (5)	Current request is a cursor request.

TDINFPGM and TDINFRPC also return this information.

Note These are new values. The old values (TDS_START_SQL and TDS_START_RPC) still work, but you should use the new values from now on.

TRAN_NAME

(O) Variable where the name of the current CICS, MVS or IMS TM transaction is returned.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-9.

Table 3-9: TDGETREQ return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_SOS (-257)	Memory shortage. The host subsystem was unable to allocate enough memory for the control block that Gateway-Library tried to create. The operation failed.

Examples

Example 1

The following code fragment illustrates the use of TDGETREQ in a program that uses the IMS TM implicit API. This example is taken from the sample program SYIPSAM1, listed in Appendix E, “Sample RPC Application for IMS TM (Implicit).”

```
/*-------------------------------------------------------------------*/
/*        WORK AREAS                                                 */
/*-------------------------------------------------------------------*/
       DCL
           01  GW_LIB_MISC_FIELDS,
               05  GWL_SPA_PTR           PTR,
               05  GWL_PROC              PTR,
               05  GWL_INIT_HANDLE       PTR,
               05  GWL_RC                FIXED BIN(31),
               05  GWL_WAIT_OPTION       FIXED BIN(31),
               05  GWL_REQ_TYPE          FIXED BIN(31),
               05  GWL_PROG_TYPE         CHAR(04) INIT('MPP '),
               05  GWL_RPC_NAME          CHAR(30);
 [rest of work areas]
/*      ---------------------------------------------------------*/
/*      set program type to MPP                                  */
/*      ---------------------------------------------------------*/
       CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
             GWL_SPA_PTR, TDS_NULL, TDS_NULL); .
 [check return code]
/*      ---------------------------------------------------------*/
/*      accept first client request                              */
/*      ---------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
              SNA_CONNECTION_NAME, SNA_SUBC); .
 [check return code; process request]
DO WHILE(MORE_MSGS);
 [prepare to send results; send reply rows]
/*-------------------------------------------------------------------*/
SEND_DONE:
/*-------------------------------------------------------------------*/
CALL TDSNDDON (GWL_PROC, GWL_RC, TDS_DONE_COUNT,
                     CTR_ROWS, TDS_ZERO, TDS_ENDRPC);
 [check return code]
       IF PARM_NR_ROWS = 0 THEN
           MORE_MSGS = FALSE;
       ELSE
          DO;
             GWL_WAIT_OPTION = TDS_TRUE;
             GWL_REQ_TYPE = 0;
             GWL_RPC_NAME = ' ';

             CALL TDGETREQ (GWL_PROC, GWL_RC,
                            GWL_WAIT_OPTION,
                            GWL_REQ_TYPE,
                            GWL_TRAN_NAME);
```

```
                SELECT (GWL_RC);
                  WHEN(TDS_OK);
                  WHEN(TDS_CONNECTION_TERMINATED)
                         MORE_MSGS = FALSE;
                  WHEN(TDS_RESULTS_COMPLETE)
                         MORE_MSGS = FALSE;
                  OTHERWISE
                      DO;
                         MORE_MSGS = FALSE;
                         CALL_NAME = 'TDGETREQ';
                         CALL DISP_ERROR;
                      END;
                 END;  /* SELECT */
              END;    /* END ELSE */

END;  /* DO WHILE MORE_MSGS = TRUE */
```

Example 2

The following code fragment illustrates the use of TDGETREQ in a program that uses the IMS TM explicit API. This example is taken from the sample program SYIXSAM1, listed in Appendix D, "Sample RPC Application for IMS TM (Explicit)."

```
 /*-------------------------------------------------------------------*/
 /*        WORK AREAS                                                 */
 /*-------------------------------------------------------------------*/
       DCL
           01  GW_LIB_MISC_FIELDS,
               05  GWL_SPA_PTR           PTR,
               05  GWL_PROC              PTR,
               05  GWL_INIT_HANDLE       PTR,
               05  GWL_RC                FIXED BIN(31),
               05  GWL_WAIT_OPTION       FIXED BIN(31),
               05  GWL_REQ_TYPE          FIXED BIN(31),
               05  GWL_PROG_TYPE         CHAR(04) INIT('MPP '),
               05  GWL_TRAN_NAME         CHAR(30);
  [rest of work areas]
 /*      ---------------------------------------------------------*/
 /*      set program type to EXPL                                   */
 /*      ---------------------------------------------------------*/
       GWL_PROG_TYPE = 'EXPL';

       CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                     GWL_SPA_PTR, TDS_NULL, TDS_NULL);
 [check return code]
  /*      ---------------------------------------------------------*/
```

```
/*      accept first client request                                  */
/*      -----------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME, SNA_SUBC);
 [check return code; process request]
DO WHILE(MORE_MSGS);
 [prepare to send results; send reply rows]
/*-------------------------------------------------------------------*/
SEND_DONE:
/*-------------------------------------------------------------------*/
       IF PARM_NR_ROWS = 0 THEN
          DO;
            MORE_MSGS = FALSE;
            GWL_SEND_DONE = TDS_ENDRPC;
          END;
       ELSE
          GWL_SEND_DONE = TDS_ENDREPLY;

       CALL TDSNDDON (GWL_PROC, GWL_RC, TDS_DONE_COUNT,
                      CTR_ROWS, TDS_ZERO, GWL_SEND_DONE);
 [check return code]
IF PARM_NR_ROWS = 0 THEN
          DO;
             GWL_WAIT_OPTION = TDS_TRUE;
             GWL_REQ_TYPE = 0;
             GWL_RPC_NAME = ' ';

             CALL TDGETREQ (GWL_PROC, GWL_RC,
                            GWL_WAIT_OPTION,
                            GWL_REQ_TYPE,
                            GWL_TRAN_NAME);
              SELECT (GWL_RC);
               WHEN(TDS_OK);
               WHEN(TDS_CONNECTION_TERMINATED)
                 DO;
                    MORE_MSGS = FALSE;
                    CALL FREE_STORAGE;
                 END;
             WHEN(TDS_RESULTS_COMPLETE)
                 DO;
                    MORE_MSGS = FALSE;
                    CALL FREE_STORAGE;
                 END;
             OTHERWISE
                 DO;
                    MORE_MSGS = FALSE;
```

```
                        CALL_NAME = 'TDGETREQ';
                        CALL DISP_ERROR;
                    END;
                END;     /* SELECT */

        END;  /* IF */

    END;    /* DO WHILE MORE_MSGS = TRUE */
```

Usage

TDGETREQ

Use this function in long-running transactions to determine whether more requests are arriving. If more requests are arriving, TDGETREQ:

- Indicates whether the request is an RPC or a language request (TDGETREQ gets this information from the login packet).
- Returns the transaction name.
- Accepts the request.

TDACCEPT cannot be used more than once in an application, and it is always used to accept the first client request received. When a long-running transaction or WFI transaction accepts multiple client requests, the transaction uses TDACCEPT to accept the first request and TDGETREQ to accept subsequent requests. Because all requests do not need to be the same type, TDGETREQ also indicates the type of request. For example, one may be an RPC, the next may be a SQL language request.

- TDGETREQ is used with WFI and explicit transactions under IMS TM and for CONVERSATIONAL-type transactions under CICS.
- TDINFRPC also returns the type of request, as well as the name of the RPC that called the current transaction.
- After a TDGETREQ call, continue coding just as you would after TDACCEPT.
- TDGETREQ follows TDSNDDON in a long-running or WFI transaction.
    - To keep the connection open after TDSNDDON returns results for the previous client request in a long-running transaction, the *CONN_OPTIONS* argument of TDSNDDON must be set to TDS_ENDREPLY. Otherwise, the conversation shuts down and TDGETREQ returns TDS_CONNECTION_TERMINATED.

- In a WFI transaction: The *CONN_OPTIONS* argument of TDSNDDON must be set to TDS_ENDRPC. TDS_ENDREPLY is not supported for IMS TM implicit transactions.

- TDGETREQ puts the transaction into RECEIVE state.
- For each new request, the transaction reads in a new login packet. The login packet indicates which type of request is being sent.
- You can use long-running transactions with both half-duplex and full-duplex connections.
- When a request is present, TDGETREQ returns TDS_OK. If no request is present, the TDGETREQ action depends on the value of the *WAIT_OPTION*:
  - When the *WAIT_OPTION* is TDS_FALSE, TDGETREQ returns TDS_CONNECTION_TERMINATED.
  - When the *WAIT_OPTION* is TDS_TRUE, TDGETREQ waits for another request; if the transaction stops, TDGETREQ returns TDS_CONNECTION_TERMINATED.

For IMS TM users

The implicit API:

- Does not support true long-running transactions. However, if an implicit IMS TM transaction is defined as WFI, it can accept multiple requests from any number of workstations for the same mainframe transaction.
- When used properly with TDGETREQ, must use a WFI transaction, or the message region that the transaction runs in must have PWFI=Y (Pseudo-Wait-For-Input) specified.

The explicit API:

- Must use the same Gateway-Library functions and parameters as CICS programs. All comments in this section apply to explicit IMS TM transactions and to CICS transactions.

See also

Related functions

- TDSNDDON on page 212

Related topics

- “Communication states” on page 19
- “The login packet” on page 52
- “Long-running transactions” on page 52

TDGETSOI

Description

Queries the Shift Out/Shift In (SO/SI) processing settings for a column or parameter.

Note This function is used with the Japanese Conversion Module (JCM).

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC        PTR,
01 RETCODE       FIXED BIN(31),
01 OBJECT_TYPE   FIXED BIN(31),
01 OBJECT_NUMBER FIXED BIN(31),
01 STRIP_SOSI    FIXED BIN(31);

CALL TDGETSOI (TDPROC, RETCODE, OBJECT_TYPE,
          OBJECT_NUMBER, STRIP_SOSI);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-10 on page 111.

OBJECT_TYPE

(I) Type of object to be checked. This argument specifies which type of object is checked by this call: a column in a return row or a return parameter.

Assign *OBJECT_TYPE* one of the following values:

TDS_OBJECT_COL(1)	Object is a column in a return row.
TDS_OBJECT_PARM (2)	Object is a return parameter.

OBJECT_NUMBER

(I) Order number of the column or parameter being checked.

If the object is a column, this is the position of the column in the row, counting from left to right. Columns are numbered sequentially with the leftmost column in a row number 1.

If the object is a return parameter, this is the number of the parameter with the value that is being checked. All parameters are counted, whether or not they are return parameters. Parameters are numbered sequentially with the first parameter number 1.

STRIP_SOSI

(O) The SO/SI processing setting being used for this column or parameter.

STRIP_SOSI returns one of the following values:

TDS_STRIP_SOSI (0)	SO/SI codes are stripped before being sent to the client. This is the default.
TDS_BLANK_SOSI (1)	SO/SI codes are converted to blanks before being sent to the client.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-10.

Table 3-10: TDGETSOI return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_FLAGS (-176)	Invalid padding option for a field.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Usage

TDGETSOI

Use TDGETSOI to determine whether SO/SI codes in double-byte character strings are stripped or converted to blanks before results are returned to the client.

- SO/SI codes are used with character datatypes to set off double-byte characters. Graphic datatypes do not use SO/SI codes.
- Replacing SO/SI codes with blanks maintains the length of the string. Otherwise, if SO/SI codes are stripped, the result length is shorter than the source length.
- For more information about Shift Out and Shift In codes, read “Character sets” on page 17 and “Processing Japanese client requests” on page 56.

See also

Related functions

- TDSETSOI on page 203

Related topics

- “Character sets” on page 17

- “Processing Japanese client requests” on page 56

TDGETUSR

Description

Gets user login information from the client.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC           PTR,
01 RETCODE          FIXED BIN(31),
01 ACCESS_CODE      CHAR(32),
01 USER_ID          CHAR(32),
01 PASSWORD         CHAR(32),
01 SERVER_NAME      CHAR(32),
01 CLIENT_CHARSET   CHAR(32),
01 NATIONAL_LANGUAGE CHAR(32),
01 SERVER_CHARSET   CHAR(32),
01 SERVER_DBCS      CHAR(32),
01 APPNAME_ID       CHAR(32);

CALL TDGETUSR' (TDPROC, RETCODE, ACCESS_CODE,
            USER_ID, PASSWORD, SERVER_NAME,
            CLIENT_CHARSET, NATIONAL_LANGUAGE,
            SERVER_CHARSET, SERVER_DBCS,
            APPNAME_ID);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-11 on page 114.

ACCESS_CODE

(I) Variable containing an access code that authorizes this application to retrieve a client password. TDGETUSR gets this information from the mainframe customization module.

USER_ID

(O) Variable where the client user ID is returned to the application. This is the user ID the client uses to log into the TRS.

PASSWORD

(O) Variable where the client password is returned to the application. This is the password the client uses to log into the TRS.

Note If an access code is required and it does not match the access code specified during mainframe customization, the PASSWORD field is set to blanks.

SERVER_NAME

(O) Variable where the name of the server specified by the client is returned. For workstation clients, this is the name of the TRS used to access this Open ServerConnect application.

CLIENT_CHARSET

(O) Variable where the name of the character set used by the client is returned. This information is provided in the client login packet.

NATIONAL_LANGUAGE

(O) Variable where the name of the national language used by the client is returned. This information is provided in the client login packet. If no national language is specified, the default is U.S. English.

SERVER_CHARSET

(O) Variable where information about the treatment of single-byte characters is returned. This value is set during customization.

If *SERVER_DBCS* indicates that double-byte character sets are not supported (*SERVER_DBCS* is NONE), *SERVER_CHARSET* returns the name of the default single-byte character set used by Gateway-Library programs. The default character set is used in the following cases:

- The client login packet does not specify a character set.
- The client login packet specifies a character set, but Gateway-Library cannot find that character set in the table of character set names.

If *SERVER_DBCS* indicates that double-byte character sets are supported (*SERVER_DBCS* is KANJI), *SERVER_CHARSET* indicates how single-byte characters are treated.

Single-byte characters can be treated as either:

LOWERCASE	Lowercase letters (roman alphabet)
KANA	Hankaku katakana (single-byte Japanese characters)

SERVER_DBCS

(O) DBCS support indicator. This value indicates whether the mainframe system is using double-byte kanji characters or only single-byte characters. TDGETUSR gets this information from the mainframe customization module.

KANJI	Double-byte characters are supported.
NONE	Double-byte characters are not supported.

APPNAME_ID

(O) Name of the client application (from the client login record). The application name is set on the client side via a dbsetlapp call, and forwarded to the mainframe by the TRS. *APPNAME_ID* is typically used to pass unique identifier information about the client application.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-11 on page 114.

Table 3-11: TDGETUSR return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates the use of TDGETUSR to verify the client login information. The program must provide an access code—TOP SECRET—for permission to access the user's password. It is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*      ------------------------------------------------------------*/
/*       accept client request                                      */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);
/*      ------------------------------------------------------------*/
/*       TDRESULT to validate kicked off via rpc request            */
/*      ------------------------------------------------------------*/
        CALL TDRESULT (GWL_PROC, GWL_RC);

        IF GWL_RC ^= TDS_PARM_PRESENT THEN
        DO;
```

```
            CALL TDRESULT_ERROR;
            GO TO END_PROGRAM;
        END;
/*       ------------------------------------------------------------*/
/*        verify user login information                              */
/*       ------------------------------------------------------------*/
        GU_ACCESS_CODE = 'TOP SECRET';

        CALL TDGETUSR (GWL_PROC, GWL_RC,
                       GU_ACCESS_CODE,
                       GU_USER_ID,
                       GU_PASSWORD,
                       GU_SERVER_NAME,
                       GU_CLIENT_CHARSET,
                       GU_NATIONAL_LANG,
                       GU_SERVER_CHARSET,
                       GU_SERVER_DBCS,
                       GU_APP_ID);

        IF GWL_RC ^= TDS_OK THEN
        DO;
            CALL TDGETUSR_ERROR;
            GO TO END_PROGRAM;
        END;
```

Usage

TDGETUSR

TDGETUSR allows a mainframe server application to retrieve client information from the login packet. This information includes:

- The user ID and password the client used to log into the TRS
- The name of the TRS through which the request is sent
- The national language used by the client
- The character set used by the client

TDGETUSR retrieves customization information from the mainframe customization module. This information includes:

- The default single-byte character set used by Gateway-Library
- How single-byte characters are treated in DBCS
- A security access code that must be entered to retrieve users' login passwords

TDGETUSR is especially useful to customers who provide their own security or accounting functions. It enables the program to uniquely identify each user of the Open ServerConnect product or application.

TDGETUSR prevents unauthorized access to a client password by requiring an access code. Unless the correct access code is specified in the *ACCESS_CODE* argument, the password is not returned to the variable specified in *PASSWORD*. In this case, TDGETUSR returns TDS_OK, but leaves *PASSWORD* blank.

Note You can deactivate this feature, allowing the program to retrieve the password without an access code.

See also

Related topics

- "Character sets" on page 17
- "Customization" on page 35
- "The login packet" on page 52

Related functions

- TDACCEPT on page 68

TDINFACT

Description

Retrieves information about Gateway-Library accounting.

Syntax

```
%INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE               FIXED BIN(31),
01 ACCOUNTING_FLAG      FIXED BIN(31),
01 ACCOUNTING_FILENAME  CHAR(08),
01 MAXNUM_TRACE_RECORDS FIXED BIN(31);

CALL TDINFACT  (IHANDLE, RETCODE, ACCOUNTING_FLAG,
            ACCOUNTING_FILENAME,
             MAXNUM_ACCT_RECORDS);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-12 on page 117.

ACCOUNTING_FLAG

(O) Accounting on/off indicator. This argument returns one of the following values:

TDS_TRUE (1)	Accounting is on.
TDS_FALSE (0)	Accounting is off.

ACCOUNTING_FILENAME

(O) Variable where the name of the accounting log is returned.

Under CICS: This is the *DATASET* name from the CICS File Control Table (FCT) entry that describes the VSAM file used for this log. As installed, this name is SYTACCT1.

Under IMS TM and MVS: Leave this field blank. IMS TM and MVS ignore this value.

MAXNUM_ACCT_RECORDS

(O) Accounting log record limit.

Under CICS: This is the maximum number of records to be allocated for this accounting file. A value of -1 indicates the system maximum.

Under IMS TM: The IMS TM system log does not have a limit.

Under MVS: Use -1. The size of the log is determined by the space allocated to the sequential file used as the MVS log.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-12.

Table 3-12: TDINFACT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Examples

The following code fragment processes a request for accounting information and returns that information to the client. This example is based on the sample program in Appendix G, "Sample Tracing and Accounting Program", which runs under CICS.

```
/*--------------------------------------------------------------------*/
TDINFACT_PROC: PROC;
/*--------------------------------------------------------------------*/
       WRKLEN1          = STG(GWL_INFACT_STATUS);
       WRKLEN2          = STG(CN_INFACT_STATUS);
       CTR_COLUMN       = CTR_COLUMN +1;
       MSG_SRVLIB_FUNC = 'TDESCRIB';
       CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN, TDSINT4,
                      WRKLEN1, GWL_INFACT_STATUS, TDS_ZERO,
                      TDS_FALSE, TDSINT4, WRKLEN1,
                      CN_INFACT_STATUS, WRKLEN2);
       IF GWL_RC NE TDS_OK THEN
       DO;
           SEND_DONE = SEND_DONE_ERROR;
           GO TO TDINFACT_EXIT;
       END;
       WRKLEN1     = STG(GWL_INFACT_FILENAME);
       WRKLEN2     = STG(CN_INFACT_FILENAME);
       CTR_COLUMN = CTR_COLUMN +1;
       CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN, TDSCHAR,
                      WRKLEN1, GWL_INFACT_FILENAME, TDS_ZERO,
                      TDS_FALSE, TDSCHAR, WRKLEN1,
                      CN_INFACT_FILENAME, WRKLEN2);
       IF GWL_RC NE TDS_OK THEN
       DO;
           SEND_DONE = SEND_DONE_ERROR;
           GO TO TDINFACT_EXIT;
       END;
       WRKLEN1     = STG(GWL_INFACT_RECORDS);
       WRKLEN2     = STG(CN_INFACT_RECORDS);
       CTR_COLUMN = CTR_COLUMN +1;
       CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN, TDSINT4,
                      WRKLEN1, GWL_INFACT_RECORDS, TDS_ZERO,
                      TDS_FALSE, TDSINT4, WRKLEN1,
                      CN_INFACT_RECORDS, WRKLEN2);
       IF GWL_RC NE TDS_OK THEN
       DO;
           SEND_DONE = SEND_DONE_ERROR;
           GO TO TDINFACT_EXIT;
       END;
       CALL TDINFACT (GWL_INIT_HANDLE, GWL_RC, GWL_INFACT_STATUS,
                      GWL_INFACT_FILENAME, GWL_INFACT_RECORDS);
```

```
        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFACT';
            GO TO TDINFACT_EXIT;
        END;
        CALL TDSNDROW (GWL_PROC, GWL_RC);
/*-------------------------------------------------------------------*/
TDINFACT_EXIT:
/*-------------------------------------------------------------------*/
        RETURN;
END TDINFACT_PROC;
```

Usage

TDINFACT

You use this function to determine whether system-wide accounting recording is on or off and, under CICS, to learn the name of the accounting log.

- This function returns accounting information recorded at the mainframe server. The TRS administrator can turn local accounting recording on and off at the TRS. Accounting at the mainframe and at the TRS are independent of each other.
- Gateway-Library accounting records the total number of TDS bytes, packets, messages, rows, requests, and cancels sent and received by Open ServerConnect from the time a TDACCEPT function initializes the TDS environment until a TDFREE is issued, and the number of seconds and milliseconds that elapsed during the conversation.
- To set accounting recording on or off, use TDSETACT.

  The accounting flag is set to off when Gateway-Library is initialized. It remains off until the program explicitly turns it on with TDSETACT, then it remains on until the program explicitly turns it off with TDSETACT. No other Gateway-Library functions turn accounting on or off.
- Accounting information is written to the accounting log after TDFREE is issued.
  - *Under CICS:* The accounting log is a VSAM ESDS file.
  - *Under IMS TM:* The accounting log is the IMS TM system log. For more information on this log, see your IMS TM documentation.
  - *Under MVS:* The accounting log is a sequential file. The DDNAME of this file is defined in SYGWXCPH.

- See the Mainframe Connect Server Option *Installation and Administration Guide* for an explanation of the Gateway-Library accounting facility, instructions for using it, and the layout of the CICS accounting log.

See also

Related functions

- TDACCEPT on page 68
- TDFREE on page 99
- TDSETACT on page 176

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDINFBCD

Description

Retrieves the length and number of decimal places for a specified decimal column or parameter.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC                  PTR,
01 RETCODE                 FIXED BIN(31),
01 OBJECT_TYPE             FIXED BIN(31),
01 OBJECT_NUMBER           FIXED BIN(31),
01 BCD_LENGTH              FIXED BIN(31),
01 BCD_NUMBER_DECIMAL_PLACES FIXED BIN(31);

CALL TDINFBCD   (TDPROC, RETCODE, OBJECT_TYPE,
            OBJECT_NUMBER, BCD_LENGTH,
              BCD_NUMBER_DECIMAL_PLACES);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-13 on page 121.

OBJECT_TYPE

(I) Object type indicator. Indicates whether the object being queried is a parameter or a column. Assign this argument one of the following values:

TDS_OBJECT_COL (1)	Object is a column in a return row.
TDS_OBJECT_PARM (2)	Object is a parameter.

OBJECT_NUMBER

(I) Number of the column or parameter.

If the object is a column, this is the position of the column in the row, counting from left to right. Columns are numbered sequentially; the leftmost column in a row is number 1.

If the object is a return parameter, this is the number of the parameter with the value that is being checked. All parameters are counted, whether or not they are return parameters. Parameters are numbered sequentially; the first parameter is number 1.

BCD_LENGTH

(O) Variable where the length of the packed decimal field is returned. When used for Sybase numeric or decimal, this is the precision of the numeric or decimal field.

BCD_NUMBER_DECIMAL_PLACES

(O) Variable where the number of decimal places in the packed decimal field is returned.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-13.

Table 3-13: TDINFBCD return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates a typical use of TDINFBCD. This program uses TDSETBCD to set the number of decimal places in the column to 2; it uses TDINFBCD to check how many decimal places are in the column. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.    */
 /*      ------------------------------------------------------------*/
DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_JC);
       DB_COLUMN_NAME_HV_PTR = ADDR(CN_JC);
       WRKLEN1               = STG(EMPLOYEE_JC);
       WRKLEN2               = STG(CN_JC);
       DB_HOST_TYPE          = TDSDECIMAL;
       DB_CLIENT_TYPE        = TDSFLT8;
       CALL DESCRIBE_COLUMN;
/*     ------------------------------------------------------------*/
/*     We must inform the Server Library how many decimal places   */
/*     are in the EMPLOYEE_JC column.                              */
/*     ------------------------------------------------------------*/
       CALL TDSETBCD (GWL_PROC, GWL_RC,
                      TDS_OBJECT_COL,
                      CTR_COLUMN,
                      TDS_DEFAULT_LENGTH,
                      GWL_SETBCD_SCALE);
/*     ------------------------------------------------------------*/
/*     Demonstrate getting decimal column information.             */
/*     ------------------------------------------------------------*/
       CALL TDINFBCD (GWL_PROC, GWL_RC,
                      TDS_OBJECT_COL,
                      CTR_COLUMN,
                      GWL_INFBCD_LENGTH,
                      GWL_INFBCD_SCALE);
```

Usage

TDINFLOG

Packed decimal data is supported in PL/1, but not in DB-Library or Client-Library. The TDINFLOG function preserves the scale and value when converting a DB-Library or Client-Library decimal value to PL/1 packed decimal data and vice versa.

Numeric and Sybase decimal are both used as Client-Library decimal datatypes.

Note Although the name of this function implies BCD data, in PL/1 this function is actually used with packed decimal data.

- A server application uses this function to retrieve length information about a column or parameter containing packed decimal information, and to retrieve information about precision and scale of a column or parameter containing Sybase numeric or decimal information.
- If this function is used to query an object that does not contain decimal values, it returns the length, but the *BCD_NUMBER_DECIMAL_PLACES* argument is ignored.
- When used to get information about a column, TDINFBCD must be preceded by a TDESCRIB call for the specified column.
- Use this function after TDINFPRM to find precision and scale of a Sybase numeric or decimal parameter.

See also

Related functions

- TDESCRIB on page 89
- TDSETBCD on page 181

TDINFLOG

Description

Determines what types of mainframe server tracing are set.

Syntax

```
%INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE              FIXED BIN(31),
01 GLOBAL_TRACE_FLAG    FIXED BIN(31),
01 API_TRACE_FLAG       FIXED BIN(31),
01 TDS_HEADER_TRACE_FLAG FIXED BIN(31),
01 TDS_DATA_TRACE_FLAG  FIXED BIN(31),
01 TRACE_ID             FIXED BIN(31),
01 TRACE_FILENAME       CHAR(08),
01 MAXNUM_TRACE_RECORDS  FIXED BIN(31);
```

```
CALL TDINFLOG (IHANDLE, RETCODE, GLOBAL_TRACE_FLAG
        API_TRACE_FLAG, TDS_HEADER_TRACE_FLAG,
        TDS_DATA_TRACE_FLAG, TRACE_ID,
        TRACE_FILENAME, MAXNUM_TRACE_RECORDS);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's TDINIT call. It corresponds to the context structure in Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-14 on page 125.

GLOBAL_TRACE_FLAG

(O) Global/specific trace indicator. This argument indicates whether tracing is on or off, and whether it is global (traces all transactions) or applies to a specific set of transactions. If tracing is set off, only errors are logged.

The *GLOBAL_TRACE_FLAG* argument returns one of the following values:

TDS_NO_TRACING (0)	All tracing is off.
TDS_TRACE_ALL_RPCS (1)	Global tracing is on.
TDS_TRACE_SPECIFIC_RPCS (2)	Specific tracing is on.
TDS_TRACE_ERRORS_ONLY (3)	Only errors are logged.

API_TRACE_FLAG

(O) API tracing on/off indicator. This is a Boolean value that indicates whether tracing is turned on or off for Gateway-Library calls. This argument returns one of the following values:

TDS_TRUE (1)	API tracing is on.
TDS_FALSE (0)	API tracing is off.

TDS_HEADER_TRACE_FLAG

(O) TDS header tracing on/off indicator. This is a Boolean value that indicates whether tracing is turned on or off for TDS headers. This argument returns one of the following values:

TDS_TRUE (1)	Header tracing is on.
TDS_FALSE (0)	Header tracing is off.

TDS_DATA_TRACE_FLAG

(O) TDS data tracing on/off indicator. This is a Boolean value that indicates whether tracing is turned on or off for TDS data. This argument returns one of the following values:

TDS_TRUE (1)	Data tracing is on.
TDS_FALSE (0)	Data tracing is off.

TRACE_ID

(O) The trace entry identifier.

Under CICS: This is the tag for the auxiliary file entry.

Under IMS TM and MVS: Leave this field blank. This argument is ignored.

TRACE_FILENAME

(O) Name of the trace/error log.

Under CICS: This is the *DATASET* name from the CICS File Control Table (FCT) entry that describes the VSAM file used for this log. As installed, this name is *SYTDLOG1*.

Under IMS TM and MVS: Leave this field blank. IMS TM and MVS ignore this value.

MAXNUM_TRACE_RECORDS

(O) Trace log record limit.

Under CICS: This is the maximum number of records that can be written to this file. A value of -1 indicates the system maximum.

Under IMS TM: The IMS TM system log does not have a limit.

Under MVS: The limit is the amount of space on the log file.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-14.

Table 3-14: TDINFLOG return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_LOG_ERROR(-258)	Attempt to write to the log file failed.

Examples

The following code fragment uses TDINFLOG to determine the tracing status at the beginning of an application. This example is taken from the sample program in Appendix C, "Sample Language Application for CICS."

```
/*     ------------------------------------------------------------*/
/*      Establish gateway environment                              */
/*      ------------------------------------------------------------*/
       CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);
/*      ------------------------------------------------------------*/
/*      turn on local tracing if not on globally or locally        */
/*      ------------------------------------------------------------*/
       CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                      GWL_INFLOG_GLOBAL,
                      GWL_INFLOG_API,
                      GWL_INFLOG_TDS_HEADER,
                      GWL_INFLOG_TDS_DATA,
                      GWL_INFLOG_TRACE_ID,
                      GWL_INFLOG_FILENAME,
                      GWL_INFLOG_TOTAL_RECS);
       IF GWL_INFLOG_GLOBAL ^= TDS_TRACE_ALL_RPCS &
          GWL_INFLOG_GLOBAL ^= TDS_TRACE_SPECIFIC_RPCS THEN
       DO;
           TRACING_SET_SW = TRACING_SET;
           CALL LOCAL_TRACING;
       END;
/*     ------------------------------------------------------------*/
/*     Accept client request                                       */
/*     ------------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME,
                      SNA_SUBC);
```

Usage

TDINFPGM

This function is used to determine whether tracing is turned on, and whether the traces are global or for specific transactions only. The following kinds of tracing are supported:

- API call tracing. This traces Gateway-Library calls under the following communication front ends:
  - CICS - Uses the CICS auxiliary trace facility.
  - IMS TM - Uses the IMS TM system log.
  - MVS - Uses a sequential file.
- TDS header tracing. This keeps track of the 8-byte TDS headers being sent to and from the mainframe server.

- TDS data tracing. This traces both incoming and outgoing TDS data.
- Trace records are written to the trace log.
- The trace log is also the error log.
- To turn tracing on or off and specify whether it is global or specific, call TDSETLOG.
- Specific tracing can be set for 1–8 transactions. To specify tracing for individual transactions, call TDSETSPT. To find out whether tracing is on for a particular transaction, call TDINFSPT. To list the transactions for which specific tracing is enabled, call TDLSTSPT.
- TDINFLOG returns trace information recorded at the mainframe server. The TRS administrator can turn tracing on and off at the TRS. Tracing at the mainframe and at the TRS are independent of each other.
- See the Mainframe Connect Server Option *Installation and Administration Guide* for an explanation of the Gateway-Library trace facility, instructions for using it, and the layout of the CICS trace log.

See also

Related functions

- TDACCEPT on page 68
- TDFREE on page 99
- TDINFSPT on page 141
- TDSETLOG on page 187
- TDSETSPT on page 206
- TDWRTLOG on page 244

Related Documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDINFPGM

Description

Retrieves information about the current client request.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC          PTR,
01 RETCODE          FIXED BIN(31),
01 TDS_VERSION      FIXED BIN(31),
```

```
01 LONGVAR_TRUNC_FLAG FIXED BIN(31),
01 ROW_LIMIT          FIXED BIN(31),
01 REMOTE_TRACE_FLAG  FIXED BIN(31),
01 USER_CORRELATOR    FIXED BIN(31),
01 DB2GW_OPTIONS      FIXED BIN(31),
01 DB2GW_PID          CHAR(8),
01 REQUEST_TYPE       FIXED BIN(31);

CALL TDINFPGM (TDPROC, RETCODE, TDS_VERSION,
          LONGVAR_TRUNC_FLAG, ROW_LIMIT,
          REMOTE_TRACE_FLAG, USER_CORRELATOR,
          DB2GW_OPTIONS, DB2GW_PID,
          REQUEST_TYPE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-16 on page 130.

TDS_VERSION

(O) Variable where the version of TDS being used is returned. The version can be any of the values listed in Table 3-15. This value must be the same as the version level specified at the client.

Table 3-15: TDS_VERSION values

TDS_VERSION2_0	FIXED BIN(31)	INIT(512)
TDS_VERSION3_4	FIXED BIN(31)	INIT(832)
TDS_VERSION4_0	FIXED BIN(31)	INIT(1024)
TDS_VERSION4_2	FIXED BIN(31)	INIT(1056)
TDS_VERSION4_6	FIXED BIN(31)	INIT(1120)
TDS_VERSION4_8	FIXED BIN(31)	INIT(1152)
TDS_VERSION4_9	FIXED BIN(31)	INIT(1168)
TDS_VERSION5_0	FIXED BIN(31)	INIT(1280)
TDS_VERSION5_1	FIXED BIN (31)	INIT(1296)

LONGVAR_TRUNC_FLAG

(O) Variable where the truncation indicator for TDSLONGVARCHAR fields is returned. It indicates what happens when any TDSLONGVARCHAR fields over 255 characters are returned to the client.

One of the following values is returned in this variable:

TDS_TRUE (1)	TDSLONGVARCHAR fields are truncated.
TDS_FALSE (0)	TDSLONGVARCHAR fields are not truncated; an error is returned instead.

If 0 is specified, it is the responsibility of the Gateway-Library programmer to determine what action is taken.

Note TDSLONGVARCHAR truncation can also be specified at the mainframe during customization. If truncation is set on at either the mainframe or the TRS, truncation occurs.

ROW_LIMIT

This argument is ignored.

REMOTE_TRACE_FLAG

(O) Variable that contains the TRS tracing indicator. This is a Boolean value that indicates whether tracing is on or off at the TRS.

One of the following values is returned in this variable:

TDS_TRUE (1)	TRS tracing is on.
TDS_FALSE (0)	TRS tracing is off.

USER_CORRELATOR

(I) Information argument. You can use this argument for any purpose.

DB2GW_OPTIONS

This argument is ignored.

DB2GW_PID

This argument is ignored.

REQUEST_TYPE

(O) Type of request to be accepted. Returns one of the following values:

TDS_LANGUAGE_EVENT (1)	Current request is a language request.
TDS_RPC_EVENT (3)	Current request is an RPC.
TDS_DYNAMIC_EVENT (4)	Current request is a dynamic SQL request.
TDS_CURSOR_EVENT (5)	Current request is a cursor request.

TDGETREQ and TDINFRPC also return this information.

Note These are new values. The old values of TDS_START_SQL and TDS_START_RPC still work, but you should use the new values from now on.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-16.

Table 3-16: TDINFPGM return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates the use of TDINFPGM to determine what kind of request was received. This example is taken from the sample program in Appendix C, “Sample Language Application for CICS.”

```
/*      Accept client request                                    */
/*      ---------------------------------------------------------*/
     CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                    SNA_CONNECTION_NAME,
                    SNA_SUBC);

/*      ---------------------------------------------------------*/
/*      ensure kicked off via language request                   */
/*      (this could be handled more reasonably by TDRESULT)      */
/*      ---------------------------------------------------------*/
     CALL TDINFPGM (GWL_PROC, GWL_RC,
```

```
                    GWL_INFPGM_TDS_VERSION,
                    GWL_INFPGM_LONGVAR,
                    GWL_INFPGM_ROW_LIMIT,
                    GWL_INFPGM_REMOTE_TRACE,
                    GWL_INFPGM_CORRELATOR,
                    GWL_INFPGM_DB2GW_OPTION,
                    GWL_INFPGM_DB2GW_PID,
                    GWL_INFPGM_TYPE_RPC);

IF GWL_INFPGM_TYPE_RPC ^= TDS_START_SQL THEN
DO;
    MSG_TEXT   = MSG_NOT_LANG;
    MSG_TEXT_L = STG(MSG_NOT_LANG);
    CALL SEND_ERROR_MESSAGE;
    GO TO END_PROGRAM;
END;
```

Usage

TDINFPGM

A server application uses this function to get information about the client request (remote program). This function can be used only by a server.

- This function returns the following information:
  - The TDS version currently in use.
  - Whether the request is an RPC, language request, cursor request, or dynamic request.
  - Whether LONG VARCHAR fields over 255 characters should be returned to the client (in truncated form).
  - Whether TRS tracing is on or off.
- The argument *USER_CORRELATOR* is available for sending site-specific information.
- TDINFPGM looks at both the TRS and mainframe customization settings to determine whether truncation occurs. The values for this procedure are outlined in the Table 3-17.

Table 3-17: TDSLONGVARCHAR truncation

TRS truncation flag	TDCUSTOM truncation flag	TDSLONGVARCHAR fields
ON	ON	Truncated
ON	OFF	Truncated
OFF	ON	Truncated
OFF	OFF	Not truncated

See also

Related documents

- Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*

TDINFPRM

Description

Retrieves parameter type, datatype, and length information about a specified RPC parameter.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC            PTR,
01 RETCODE           FIXED BIN(31),
01 PARM_ID           FIXED BIN(31),
01 DATATYPE          FIXED BIN(31),
01 ACTUAL_DATA_LENGTH FIXED BIN(31),
01 MAX_DATA_LENGT     FIXED BIN(31),
01 PARM_STATU         FIXED BIN(31),
01 PARM_NAME          CHAR(30),
01 PARM_NAME_LENGTH   FIXED BIN(31),
01 USER_DATATYP       FIXED BIN(31);

CALL TDINFPRM   (TDPROC, RETCODE, PARM_ID, DATATYPE,
          ACTUAL_DATA_LENGTH, MAX_DATA_LENGTH,
          PARM_STATUS, PARM_NAME,
           PARM_NAME_LENGTH, USER_DATATYPE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-18 on page 134.

PARM_ID

(I) Number of the parameter with the information that is requested. Parameters are numbered sequentially; the first parameter is number 1.

DATATYPE

(O) Variable where the Open Client datatype of the parameter is returned. The datatype is specified by the client.

ACTUAL_DATA_LENGTH

(O) Variable where the actual length of the parameter data is returned. For TDSVARYCHAR, TDSVARYBIN, and TDSVARYGRAPHIC parameters, this value does not include the 2 bytes for the “LL” length specification.

MAX_DATA_LENGTH

(O) Variable where the maximum length allowed for the parameter's data is returned. This value is specified by the client in the parameter definition. For TDSVARYCHAR, TDSVARYBIN, and TDSVARYGRAPHIC parameters, this value does not include the 2 bytes for the "LL" length specification.

PARM_STATUS

(O) Variable where the parameter's status is returned. This argument indicates whether or not the named parameter is a return parameter. It returns one of the following values, depending on the TDS version you are using:

TDS 4.6:

TDS_INPUT_VALUE (0)	Parameter is not a return parameter.
TDS_RETURN_VALUE (1)	Parameter is a return parameter.

TDS 5.0:

TDS_INPUT_VALUE_NULLABLE (32)	Parameter is a nullable non-return parameter.
TDS_RETURN_VALUE_NULLABLE (33)	Parameter is a nullable return parameter.

The client specifies the value of this argument.

PARM_NAME

(O) Variable where the name of the incoming parameter is stored. This is the name given to the parameter by the client.

PARM_NAME_LENGTH

(O) Variable where the length of the parameter name is returned. The name length is specified by the client when the RPC is sent.

USER_DATATYPE

(O) Variable where the user-assigned datatype for this parameter is stored. This argument is used for return parameters only.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-18 on page 134.

Table 3-18: TDINFPRM return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.

Return value	Meaning
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_NO_PARM_PRESENT (103)	No incoming parameters present. TDRCVPRM cannot retrieve a parameter because no more parameters were accepted. The operation failed.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates a typical use of TDINFPRM. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
 /*-----------------------------------------------------------------*/
GET_PARMS:
 /*-----------------------------------------------------------------*/

 /*      ---------------------------------------------------------*/
 /*      get return parameter information                          */
 /*      ---------------------------------------------------------*/
       GWL_INFPRM_ID = 1;
       CALL GET_PARM_INFO;

       (IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE AND
       IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE_NULLABLE) THEN
       DO;

           CALL TDINFPRM_NOT_RETURN_PARM_ERROR;
           GO TO END_PROGRAM;
       END;

       GWL_SETPRM_USER_DATA = GWL_INFPRM_USER_DATA;
```

```
      GWL_SETPRM_ID         = GWL_INFPRM_ID;
      GWL_SETPRM_DATA_L     = GWL_INFPRM_DATA_L;
      GWL_SETPRM_TYPE       = GWL_INFPRM_TYPE;

/*    ------------------------------------------------------------*/
/*    get department id parameter number from known name          */
/*    ------------------------------------------------------------*/
      GWL_INFPRM_NAME   = '@parm2';
      GWL_INFPRM_NAME_L = 6;

      CALL TDLOCPRM (GWL_PROC,
                     GWL_INFPRM_ID,
                     GWL_INFPRM_NAME,
                     GWL_INFPRM_NAME_L);

/*    ------------------------------------------------------------*/
/*    get department parameter information                        */
/*    ------------------------------------------------------------*/
      CALL GET_PARM_INFO;

      IF GWL_INFPRM_TYPE ^= TDSVARYCHAR THEN
      DO;
          CALL TDINFPRM_NOT_CHAR_PARM_ERROR;
          GO TO END_PROGRAM;
      END;

/*    ------------------------------------------------------------*/
/*    get department parameter data                               */
/*    ------------------------------------------------------------*/
      CALL TDRCVPRM (GWL_PROC, GWL_RC,
                     GWL_INFPRM_ID,
                     PARM_DEPT,
                     GWL_INFPRM_TYPE,
                     GWL_INFPRM_MAX_DATA_L,
                     GWL_RCVPRM_DATA_L);
/*------------------------------------------------------------------*/
GET_PARM_INFO: PROC;
/*------------------------------------------------------------------*/
      CALL TDINFPRM (GWL_PROC, GWL_RC,
                     GWL_INFPRM_ID,
                     GWL_INFPRM_TYPE,
                     GWL_INFPRM_DATA_L,
                     GWL_INFPRM_MAX_DATA_L,
                     GWL_INFPRM_STATUS,
                     GWL_INFPRM_NAME,
                     GWL_INFPRM_NAME_L,
```

```
                                GWL_INFPRM_USER_DATA);

            RETURN;

    END GET_PARM_INFO;
```

Usage

TDINFRPC

A server application uses this function to retrieve type and length information about a parameter before it retrieves it. This can be any supported type of parameter including cursor parameters.

- An application can request information about parameters in any order, by specifying the parameter in the *PARM_ID* argument.
- The maximum number of parameters that can be retrieved is 255.
- Unless you already know the length and datatype of the incoming parameter, you must issue a TDINFPRM call before each TDRCVPRM call. TDRCVPRM needs to know the appropriate datatype and length to properly set up for the incoming data.
- An application uses TDINFPRM only when the client request is an RPC or a cursor command. Language requests do not have parameters.
- A server program can modify the data length of a parameter by calling TDSETPRM before passing results back to the client.
- Each parameter has an actual data length and a maximum data length. For standard fixed-length datatypes that do not allow nulls, both lengths are the same. For variable-length fields, the lengths may vary.

  For example, a TDSVARYCHAR parameter with a declared length of 30 may have data that is only 10 bytes long. In this case, the parameter's actual data length is 10 and its maximum data length is 30.
- A value of TDSVOID in the *DATATYPE* argument indicates a NULL parameter (for example, an empty parameter).

See also

Related functions

- TDACCEPT on page 68
- TDNUMPRM on page 160
- TDRCVPRM on page 162
- TDSETPRM on page 194

TDINFRPC

Description

Returns information about the current client request.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC       PTR,
01 RETCODE      FIXED BIN(31),
01 REQUEST_TYPE FIXED BIN(31),
01 RPC_NAME     CHAR(n),
01 COMM_STATE   FIXED BIN(31);

CALL TDINFRPC (TDPROC, RETCODE, REQUEST_TYPE,
               RPC_NAME, COMM_STATE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-19 on page 139.

REQUEST_TYPE

(O) Type of request to be accepted. Returns one of the following values:

TDS_LANGUAGE_EVENT (1)	Current request is a language request.
TDS_RPC_EVENT (3)	Current request is an RPC.
TDS_DYNAMIC_EVENT (4)	Current request is a dynamic SQL request.
TDS_CURSOR_EVENT (5)	Current request is a cursor request.

TDINFPGM and TDGETREQ also return this information.

Note These are new values. The old values (TDS_START_SQL and TDS_START_RPC) still work, but you should use the new values from now on.

RPC_NAME

(O) Variable where the name of the current client RPC is returned. If the client request is not an RPC, this field contains blanks.

COMM_STATE

(O) Variable where the current communication state of the mainframe transaction is stored. *COMM_STATE* is one of the following values:

TDS_RESET (0)	Client/server conversation for this transaction ended. If the current transaction is running under CICS or uses the IMS TM explicit API, the transaction should exit as soon as possible. If the current transaction is a WFI transaction using the IMS TM implicit API, the transaction can accept another client request by calling TDGETREQ.
TDS_SEND (1)	Transaction is in SEND state.
TDS_RECEIVE (2)	Transaction is in RECEIVE state.

TDSTATUS also returns this information.

See "Communication states" on page 19 for an explanation of SEND and RECEIVE states.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-19 on page 139.

Table 3-19: TDINFRPC return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.

Usage

TDINFSPT

Use TDINFRPC in long-running transactions to determine:

- The type of client request currently being processed
- The name of the current client request, if the request is an RPC
- Whether the transaction is in the correct communication state for retrieving the next request (issuing TDGETREQ)

Long-running transactions use TDGETREQ to retrieve each request that follows the first request. TDGETREQ returns the request type and transaction name for each client request it accepts.

An application program can call TDINFRPC at any point in the program to retrieve information about the RPC or communication state.

The Gateway-Library function, TDSTATUS, also returns the communication state in addition to TDS status information and standard communication error codes. Call TDSTATUS when all incoming parameters are retrieved or after all results are sent. Call TDINFRPC to learn the current communication state at all other times.

- To change the communication state:
  - From RECEIVE state to SEND state, call TDRESULT. This shifts the transaction into SEND state and cancels the current request.
  - From SEND state to RECEIVE state, call TDSNDDON. This indicates that all results are sent and processing for the current request ended.
- TDINFRPC is not a required call.

For IMS TM users

When using the IMS TM implicit API:

- If *COMM_STATE* is TDS_RESET and this is a WFI transaction, the transaction can either end or call TDGETREQ to accept another client request.

When using the IMS TM explicit API:

- If *COMM_STATE* is TDS_RESET, the transaction should exit as soon as possible.
- See "Long-running transactions" on page 52 for information about coding long-running transactions under IMS TM.

See also

Related functions

- TDGETREQ on page 103
- TDINFPGM on page 127
- TDSNDDON on page 212
- TDSTATUS on page 235

Related topics

- "Communication states" on page 19
- "Long-running transactions" on page 52

TDINFSPT

Description

Indicates whether tracing is on or off for a specified transaction.

Syntax

```
%INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE              FIXED BIN(31),
01 TRACE_STATUS         FIXED BIN(31),
01 TRACE_OPTION         FIXED BIN(31),
01 TRANSACTION_ID       CHAR(n),
01 TRANSACTION_ID_LENGTH FIXED BIN(31);

CALL TDINFSPT (IHANDLE, RETCODE, TRACE_STATUS,
          TRACE_OPTION, TRANSACTION_ID,
          TRANSACTION_ID_LENGTH);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-20 on page 142.

TRACE_STATUS

(O) Variable where the trace indicator for the specified transaction is returned. This is a Boolean value that indicates whether tracing is on or off for the transaction specified in this function.

This argument returns one of the following values:

TDS_TRUE (1)	Tracing is on for this transaction.
TDS_FALSE (0)	Tracing is off for this transaction.

TRACE_OPTION

(O) Variable where the type of tracing enabled for the specified transaction is returned. This argument returns one of the following values:

TDS_SPT_API_TRACE (0x08)	All Gateway-Library calls are traced.
TDS_SPT_ERRLOG (0x02)	Error log recording is enabled.
TDS_SPT_TDS_DATA (0x01)	TDS packet-tracing recording is enabled.

TRANSACTION_ID

(I) Mainframe transaction identifier of the transaction for which the trace status is requested.

Under CICS: This is the *TRANSID* from the CICS Program Control Table (PCT).

Under IMS TM: This is the transaction name defined when the system is generated.

Under MVS: This is the APPC transaction name defined in the transaction profile.

TRANSACTION_ID_LENGTH

(O) Variable where the length of the *TRANSACTION_ID* is returned. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

Under CICS: For CICS Version 1.7, this value is always 4 or less. For later versions, it is the actual length of the transaction ID, which can be greater than 4.

Under IMS TM: This value is always 8 or less.

Under MVS: This is the APPC transaction name defined in the transaction profile. This value is normally 8 or less.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-20.

Table 3-20: TDINFSPT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid IHANDLE specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Examples

The following code fragment illustrates the use of TDINFSPT to determine whether tracing is enabled for a particular transaction. This example is taken from the sample program in Appendix G, “Sample Tracing and Accounting Program” which runs under CICS.

```
 /*-------------------------------------------------------------*/
GET_TRACE_STATUS: PROC;
 /*-------------------------------------------------------------*/
```

```
/*      ------------------------------------------------------*/
/*      find global status                                    */
/*      ------------------------------------------------------*/
     CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                    GWL_INFLOG_GLOBAL,
                    GWL_INFLOG_API,
                    GWL_INFLOG_HEADER,
                    GWL_INFLOG_DATA,
                    GWL_INFLOG_TRACEID,
                    GWL_INFLOG_FILENAME,
                    GWL_INFLOG_RECORDS);
/*      ------------------------------------------------------*/
/*      if any error, then assume tracing disabled            */
/*      ------------------------------------------------------*/
     IF GWL_RC NE TDS_OK THEN
     DO;
         SEND_DONE       = SEND_DONE_ERROR;
         MSG_SRVLIB_FUNC = 'TDINFLOG';
         GO TO GET_TRACE_STATUS_EXIT;
     END;

/*      ------------------------------------------------------*/
/*      if global tracing on, then tracing enabled            */
/*      ------------------------------------------------------*/
     IF GWL_INFLOG_GLOBAL = TDS_TRACE_ALL_RPCS THEN
     DO;
         TRACING = TRACING_ON;
         GO TO GET_TRACE_STATUS_EXIT;
     END;
/*      ------------------------------------------------------*/
/*      if error logging on, then tracing enabled             */
/*      ------------------------------------------------------*/
     IF GWL_INFLOG_GLOBAL = TDS_TRACE_ERRORS_ONLY THEN
     DO;

         TRACING = TRACING_ON;
         GO TO GET_TRACE_STATUS_EXIT;
     END;


/*      ------------------------------------------------------*/
/*      if specific tracing not on, then no tracing on        */
/*      ------------------------------------------------------*/
     IF GWL_INFLOG_GLOBAL NE TDS_TRACE_SPECIFIC_RPCS THEN
     DO;
```

```
            GO TO GET_TRACE_STATUS_EXIT;
        END;
 /*       ------------------------------------------------------*/
 /*       specific tracing is on, see if on for this transaction*/
 /*       ------------------------------------------------------*/
        WRKLEN1 = STG(WRK_RPC);

        CALL TDINFSPT (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFSPT_STATUS,
                       GWL_INFSPT_OPTIONS,
                       WRK_RPC,
                       WRKLEN1);

        IF (GWL_RC NE TDS_OK AND GWL_RC NE TDS_ENTRY_NOT_FOUND) THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINSPT';
            GO TO GET_TRACE_STATUS_EXIT;
        END;

        IF GWL_INFSPT_STATUS = TDS_TRUE THEN
        DO;
            TRACING = TRACING_ON;
        END;

 /*-------------------------------------------------------------*/
GET_TRACE_STATUS_EXIT:
 /*-------------------------------------------------------------*/
        RETURN;

END GET_TRACE_STATUS;
```

Usage

TDINFSPT

This function indicates whether tracing for a specified transaction is currently on or off.

- Transaction-level tracing occurs when TDSETLOG sets the global trace flag to TDS_TRACE_SPECIFIC_RPCS and sets on one or more types of tracing (for example, API tracing or header tracing). When the global trace flag is set to TDS_TRACE_ALL_RPCS, all transactions are traced, whether they have individual tracing turned on or not.

  Use TDINFLOG to determine the setting of the global trace flag and to learn what types of tracing are currently enabled. Use TDSETLOG to specify those settings.

- Transaction-level tracing can be enabled for up to eight transactions at a time.
- To learn how to set tracing on or off for a specified transaction, see TDSETSPT on page 206.
- To learn how to get a list of all transactions for which tracing is currently enabled, see TDLSTSPT on page 155.
- TDINFSPT governs tracing at the mainframe server. The TRS administrator can turn tracing on and off at the TRS. Tracing at the mainframe server and at the TRS are independent of each other.
- See the Mainframe Connect Server Option *Installation and Administration Guide* for an explanation of the Gateway-Library tracing facility, instructions for using it, and the layout of the trace log.

See also

Related functions

- TDINFLOG on page 123
- TDLSTSPT on page 155
- TDSETLOG on page 187
- TDSETSPT on page 206
- TDWRTLOG on page 244

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDINFUDT

Description

Retrieves information about the client-defined datatype for a column.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC       PTR,
01 RETCODE      FIXED BIN(31),
01 COLUMN_NUMBER FIXED BIN(31),
01 USER_DATATYPE FIXED BIN(31);

CALL TDINFUDT (TDPROC, RETCODE, COLUMN_NUMBER,
               USER_DATATYPE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-21.

COLUMN_NUMBER

(I) Number of the column with the datatype that is being queried. Columns are numbered sequentially; the first column in a row is number 1.

USER_DATATYPE

(O) Variable where the user-defined datatype is returned. This can be any datatype assigned to the column by a client.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-21.

Table 3-21: TDINFUDT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates a typical use of TDINFUDT. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
 /*-------------------------------------------------------------------*/
SETUP_REPLY_COLUMNS:
 /*-------------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_FNM);
      DB_COLUMN_NAME_HV_PTR  = ADDR(CN_FNM);
      WRKLEN1                = STG(EMPLOYEE_FNM)-2;
      WRKLEN2                = STG(CN_FNM);
      DB_HOST_TYPE           = TDSVARYCHAR;
      DB_CLIENT_TYPE         = TDSVARYCHAR;
      CALL DESCRIBE_COLUMN;

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

```
/*      Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR) */
/*      to DBCHAR.                                                  */
/*      ------------------------------------------------------------*/
     DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_LNM);
     DB_COLUMN_NAME_HV_PTR  = ADDR(CN_LNM);
     WRKLEN1                = STG(EMPLOYEE_LNM)-2;
     WRKLEN2                = STG(CN_LNM);
     DB_HOST_TYPE           = TDSVARYCHAR;
     DB_CLIENT_TYPE         = TDSCHAR;
     CALL DESCRIBE_COLUMN;

     DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_ED);
     DB_COLUMN_NAME_HV_PTR  = ADDR(CN_ED);
     WRKLEN1                = STG(EMPLOYEE_ED);
     WRKLEN2                = STG(CN_ED);
     DB_HOST_TYPE           = TDSINT2;
     DB_CLIENT_TYPE         = TDSINT2;
     CALL DESCRIBE_COLUMN;

/*      ------------------------------------------------------------*/
/*      Get the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
     CALL TDINFUDT (GWL_PROC, GWL_RC,
                    CTR_COLUMN,
                    GWL_INFUDT_USER_TYPE);

/*      ------------------------------------------------------------*/
/*      Set the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
     CALL TDSETUDT (GWL_PROC, GWL_RC,
                    CTR_COLUMN,
                    GWL_INFUDT_USER_TYPE);
```

Usage

TDINFUDT

Use this function to determine the datatype defined for a column by the client. When your application returns results to the client, it can specify the user-defined datatype for that column with the function TDSETUDT.

- The user-defined datatype is a tag associated with a column by the client. It is not the TDS datatype of the column, which is specified in the TDESCRIB call.
- You can query and set the user-defined datatype for a return parameter with TDINFPRM and TDSETPRM.

See also

Related functions

- TDINFPRM on page 133
- TDSETPRM on page 194
- TDSETUDT on page 210

TDINIT

Description

Initializes the TDS environment for a connection.

Syntax

```
%INCLUDE SYGWPLI;

01 RETCODE    FIXED BIN(31),
01 IHANDLE    PTR;
For CICS: CALL TDINIT (DFHEIBLK, RETCODE, IHANDLE);
For IMS TM: CALL TDINIT  (IO_PCB, RETCODE, IHANDLE);
For MVS: CALL TDINIT (DUMMY, GWL_RC, GWL_INIT_HANDLE);
```

Note MVS applications should pass a null pointer to TDINIT. *DUMMY* should be declared as: *DCL DUMMY PTR;*

Parameters

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-22.

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. All subsequent tracing and accounting functions must specify this same value in their *IHANDLE* argument. It corresponds to the context structure in Open Client Client-Library.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-22.

Table 3-22: TDINIT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONTROL_NOTLOADED (-260)	Cannot load the customization module. This module is necessary for Gateway-Library operation.

Return value	Meaning
TDS_GWLIB_BAD_VERSION (-16)	The program version you are using is newer than the version of the Gateway-Library phase in use.
TDS_GWLIB_UNAVAILABLE (-15)	Could not load SYGWCICS (the Gateway-Library phase).
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_SOS (-257)	Memory shortage. The host subsystem was unable to allocate enough memory for the control block that Gateway-Library was trying to create. The operation failed.

Examples

Example 1

The following code fragment illustrates the use of TDINIT, TDACCEPT, TDSNDDON, and TDFREE at the beginning and end of a Gateway-Library program. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
 /*-------------------------------------------------------------------*/
INITIALIZE_PROGRAM:
 /*-------------------------------------------------------------------*/

 /*      ---------------------------------------------------------*/
 /*      reset db2 error handlers                                 */
 /*      ---------------------------------------------------------*/
       EXEC SQL WHENEVER SQLWARNING CONTINUE;
       EXEC SQL WHENEVER SQLERROR   CONTINUE;
       EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/*      ---------------------------------------------------------*/
/*      establish gateway environment                            */
/*      ---------------------------------------------------------*/
       CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);

/*      ---------------------------------------------------------*/
/*      accept client request                                    */
/*      ---------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME,
                      SNA_SUBC);
/*-------------------------------------------------------------------*/
/*  (BODY OF PROGRAM)                                                */
/*-------------------------------------------------------------------*/
       GO TO END_PROGRAM;
/*-------------------------------------------------------------------*/
```

```
 END_PROGRAM:
/*-------------------------------------------------------------------*/
        IF SEND_DONE = SEND_DONE_OK THEN
            WRK_DONE_STATUS = TDS_DONE_COUNT;

        ELSE
        DO;
            WRK_DONE_STATUS  = TDS_DONE_ERROR;
            PARM_RETURN_ROWS = 0;
        END;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       WRK_DONE_STATUS,
                       PARM_RETURN_ROWS,
                       TDS_ZERO,
                       TDS_ENDRPC);

        CALL TDFREE (GWL_PROC, GWL_RC);
        EXEC CICS RETURN;

END SYSAMP1;
```

Example 2

The following code fragment illustrates the use of TDINIT, TDSETPT, and TDACCEPT at the beginning of a Gateway-Library program. This example is taken from the sample program in Appendix D, "Sample RPC Application for IMS TM (Explicit)."

```
 /*     ---------------------------------------------------------*/
 /*      establish gateway environment                           */
 /*      ---------------------------------------------------------*/
       CALL TDINIT (P_PCBTERM, GWL_RC, GWL_INIT_HANDLE);
 [check return code]
 /*      ---------------------------------------------------------*/
 /*      set program type to EXPL                                */
 /*      ---------------------------------------------------------*/
       GWL_PROG_TYPE = 'EXPL';

       CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                     GWL_SPA_PTR, TDS_NULL, TDS_NULL);
 [check return code]
 /*      ---------------------------------------------------------*/
 /*      accept client request                                   */
 /*      ---------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME,
                      SNA_SUBC);
```

```
[check return code]
```

Usage

TDINIT

TDINIT initializes the TDS environment for a new client/server connection, preparing the connection for data transfer between the Gateway-Library transaction and the remote client.

- This function must be the first Gateway-Library function called in a server program, and can be called only once for a given connection.
- TDINIT is also the first function called in a mixed client/server program. See the example in Appendix F, "Sample Mixed-Mode Application."
- The first TDINIT argument is the address of the communication I/O block.

  Under CICS: This is the EXEC Interface Block (EIB). You must code "DFHEIBLK" exactly as shown in the first call in the Syntax example.

  Under IMS TM: This is the I/O Program Communications Block. You must code "IO_PCB" exactly as shown in the second call in the Syntax example.

  Under MVS: Pass a null pointer. MVS does not use it.

  Note For Open ServerConnect, the conversation is always initiated by the client program. Gateway-Library programs do not initiate conversations.

- You customize your Gateway-Library environment when Open ServerConnect is installed. TDINIT loads the customization module. If it cannot load that module, TDINIT returns TDS_CONTROL_NOTLOADED. Without this module, Gateway-Library programs cannot be used.

  During customization, the national language and default character sets used at the mainframe are specified. A Gateway-Library program can retrieve customization information with TDGETUSR.

For Japanese users

The Japanese Conversion Module (JCM) processes Japanese requests. The JCM is an option available with Open ServerConnect which must be installed and defined to your mainframe system.

- TDINIT loads the JCM. If it cannot load that module, TDINIT does not return an error code. However, when a client request specifies a double-byte character set in the login packet, TDACCEPT returns TDS_CHARSET_NOTLOADED.

- See "Character sets" on page 17 and "Processing Japanese client requests" on page 56 for more information about using Gateway-Library with Japanese characters.

See also

Related functions

- TDACCEPT on page 68
- TDFREE on page 99
- TDGETUSR on page 112

Related topics

- "Character sets" on page 17
- "Customization" on page 35
- "Processing Japanese client requests" on page 56

TDLOCPRM

Description

Returns the ID number of a parameter when the parameter name is received.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC          PTR,
01 PARM_ID         FIXED BIN(31),
01 PARM_NAME       CHAR(n),
01 PARM_NAME_LENGTH FIXED BIN(31);

CALL TDLOCPRM (TDPROC, PARM_ID, PARM_NAME,
          PARM_NAME_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

PARM_ID

(O) Variable where the number of the named parameter is returned. Parameters are numbered sequentially; the ID of the first parameter is 1. If a 0 is returned here, TDLOCPRM could not find a parameter with the specified name.

PARM_NAME
(I) The name associated with the desired parameter. This name corresponds to the parameter name in the Open Client DB-Library dbrpcparam routine.

PARM_NAME_LENGTH
(I) The actual length of the *PARM_NAME*.

Return value

This function has no *RETCODE* argument. It returns the parameter ID in the *PARM_ID* argument, or a 0 if it finds no parameter with the specified name.

Examples

The following code fragment illustrates a typical use of TDLOCPRM. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*------------------------------------------------------------- */
        GET_PARMS:
/*------------------------------------------------------------- */

/*      ------------------------------------------------------- */
/*      get return parameter information
/*      ------------------------------------------------------- */
       GWL_INFPRM_ID = 1;
       CALL GET_PARM_INFO;

       (IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE AND
       IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE_NULLABLE) THEN
       DO;
           CALL TDINFPRM_NOT_RETURN_PARM_ERROR;
           GO TO END_PROGRAM;
       END;

       GWL_SETPRM_USER_DATA = GWL_INFPRM_USER_DATA;
       GWL_SETPRM_ID        = GWL_INFPRM_ID;
       GWL_SETPRM_DATA_L    = GWL_INFPRM_DATA_L;
       GWL_SETPRM_TYPE      = GWL_INFPRM_TYPE;

/*     ------------------------------------------------------------*/
/*     get department id parameter number from known name          */
/*     ------------------------------------------------------------*/
       GWL_INFPRM_NAME   = '@parm2';
       GWL_INFPRM_NAME_L = 6;

       CALL TDLOCPRM (GWL_PROC,
                      GWL_INFPRM_ID,
                      GWL_INFPRM_NAME,
                      GWL_INFPRM_NAME_L);

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

```
/*      get department parameter information                        */
/*      ------------------------------------------------------------*/
        CALL GET_PARM_INFO;

        IF GWL_INFPRM_TYPE ^= TDSVARYCHAR THEN
        DO;
            CALL TDINFPRM_NOT_CHAR_PARM_ERROR;
            GO TO END_PROGRAM;
        END;



/*      ------------------------------------------------------------*/
/*      get department parameter data                               */
/*      ------------------------------------------------------------*/
        CALL TDRCVPRM (GWL_PROC, GWL_RC,
                       GWL_INFPRM_ID,
                       PARM_DEPT,
                       GWL_INFPRM_TYPE,
                       GWL_INFPRM_MAX_DATA_L,
                       GWL_RCVPRM_DATA_L);
/*------------------------------------------------------------------*/
END_OF_QUERY:
/*------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      close cursor                                                */
/*      ------------------------------------------------------------*/
        EXEC SQL CLOSE ECURSOR;

/*      ------------------------------------------------------------*/
/*      update return parameter with nr of rows fetched             */
/*      ------------------------------------------------------------*/
        CALL TDSETPRM (GWL_PROC, GWL_RC,
                       GWL_SETPRM_ID,
                       GWL_SETPRM_TYPE,
                       GWL_SETPRM_DATA_L,

                       PARM_RETURN_ROWS,
                       GWL_SETPRM_USER_DATA);

        GO TO END_PROGRAM;
```

Usage

TDLOCPRM

A server application uses this function to determine the ID of a parameter with a name that is known.

- If no parameter matching the specified name is found, this function returns 0 in the *PARM_ID* argument.

See also

Related functions

- TDINFPRM on page 133
- TDRCVPRM on page 162

Related documents

- Open Client DB-Library *Reference Manual* (dbrpcparam)

TDLSTSPT

Description

Lists transactions for which tracing is enabled.

Syntax

```
%INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE              FIXED BIN(31),
01 TRACE_TABLE_LIST(8) CHAR(08);

CALL TDLSTSPT (IHANDLE, RETCODE, TRACE_TABLE_LIST);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-23.

TRACE_TABLE_LIST

(O) An array listing the contents of the trace table. Each element of this array, *TRANSID_n*, returns the transaction ID of a transaction for which specific tracing is currently enabled.

Under CICS: This is the TRANSID from the CICS Program Control Table (PCT).

Under IMS TM: This is the transaction name defined when the system is generated.

Under MVS: This is the APPC transaction name of the MVS transaction.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-23.

Table 3-23: TDLSTSPT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Examples

The following code fragment illustrates the use of TDLSTSPT to determine which transactions have tracing enabled. It returns the transaction IDs to the caller. This example is taken from the sample program in Appendix G, "Sample Tracing and Accounting Program," which runs under CICS.

```
/*-------------------------------------------------------------*/
   INITIALIZE_PROGRAM:
/*-------------------------------------------------------------*/
      CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);

      CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
      SNA_CONNECTION_NAME, SNA_SUBC);

      CALL TDINFRPC (GWL_PROC, GWL_RC, GWL_REQ_TYPE, GWL_RPC_NAME,
                     GWL_COMM_STATE);

      IF GWL_RC NE TDS_PARM_PRESENT THEN
      DO;
          SEND_DONE       = SEND_DONE_ERROR;
          MSG_SRVLIB_FUNC = 'TDINFRPC';
          GO TO END_PROGRAM;
      END;
```

```
/*-------------------------------------------------------------*/
   GET_PARM:
/*-------------------------------------------------------------*/
      CALL TDRCVPRM (GWL_PROC, GWL_RC, GWL_RCVPRM_ID,
                     PARM_REQUEST, TDSCHAR,
                     GWL_RCVPRM_MAX_DATA_L, GWL_RCVPRM_DATA_L);
      IF PARM_REQUEST = PARM_REQUEST_INFACT THEN
          CALL TDINFACT_PROC;

      ELSE IF PARM_REQUEST = PARM_REQUEST_LSTSPT THEN
          CALL TDLSTSPT_PROC;

/*-------------------------------------------------------------*/
    TDLSTSPT_PROC: PROC;
/*-------------------------------------------------------------*/
      WRKLEN1          = STG(WRK_TRANID);
      WRKLEN2          = STG(CN_LSTSPT_TRANID);
      CTR_COLUMN       = CTR_COLUMN +1;
      MSG_SRVLIB_FUNC = 'TDESCRIB';

      CALL TDESCRIB (GWL_PROC, GWL_RC,
                     CTR_COLUMN,
                     TDSCHAR,
                     WRKLEN1,
                     WRK_TRANID,
                     TDS_ZERO,
                     TDS_FALSE,
                     TDSCHAR,
                     WRKLEN1,
                     CN_LSTSPT_TRANID,
                     WRKLEN2);

      IF GWL_RC NE TDS_OK THEN
      DO;
          SEND_DONE = SEND_DONE_ERROR;
          GO TO TDLSTSPT_EXIT;
      END;

/*      ---------------------------------------------------*/
/*      find global status                                 */
/*      ---------------------------------------------------*/
      CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
GWL_INFLOG_GLOBAL,
GWL_INFLOG_API,
GWL_INFLOG_HEADER,
GWL_INFLOG_DATA,
```

```
    GWL_INFLOG_TRACEID,
    GWL_INFLOG_FILENAME,
    GWL_INFLOG_RECORDS);
    /*      ------------------------------------------------------*/
    /*      if any error, then assume tracing disabled            */
    /*      ------------------------------------------------------*/
          IF GWL_RC NE TDS_OK THEN
          DO;
              SEND_DONE       = SEND_DONE_ERROR;
              MSG_SRVLIB_FUNC = 'TDINFLOG';
              GO TO TDLSTSPT_EXIT;
          END;
    /*      ------------------------------------------------------*/
    /*      if not specific tracing, then return nothing          */
    /*      ------------------------------------------------------*/
          IF GWL_INFLOG_GLOBAL NE TDS_TRACE_SPECIFIC_RPCS THEN
              GO TO TDLSTSPT_EXIT;
/*        ------------------------------------------------------*/
/*        return rows                                           */
/*        ------------------------------------------------------*/
          CALL TDLSTSPT (GWL_INIT_HANDLE, GWL_RC,
                         GWL_LSTSPT_LIST(1));
          IF GWL_RC NE TDS_OK THEN
          DO;
              SEND_DONE       = SEND_DONE_ERROR;
              MSG_SRVLIB_FUNC = 'TDLSTSPT';
          END;
          DOROW: DO WRK_LSTSPT_SS = 1 TO 8;
              WRK_TRANID = GWL_LSTSPT_LIST(WRK_LSTSPT_SS);
              CALL TDSNDROW (GWL_PROC, GWL_RC);
              IF GWL_RC NE TDS_OK THEN
              DO;
                  SEND_DONE       = SEND_DONE_ERROR;
                  MSG_SRVLIB_FUNC = 'TDSNDROW';
                  LEAVE DOROW;
              END;
              CTR_ROWS = CTR_ROWS +1;
          END;
/*------------------------------------------------------------*/
  TDLSTSPT_EXIT:
/*------------------------------------------------------------*/
          RETURN;
END TDLSTSPT_PROC;
```

Usage

TDLSTSPT

TDLSTSPT lists the transactions for which specific tracing is enabled. Transaction-level tracing can be enabled for up to eight transactions.

- A blank indicates that no more transactions have tracing enabled. For example, if the first four elements in the array return transaction names, and the fifth element returns a blank, you know that tracing is enabled for four transactions only, and that elements six through eight return blanks.
- Transaction-level tracing occurs when the global trace flag is set off (TDS_FALSE) by TDSETLOG and one or more types of tracing are enabled. When the global trace flag is set on TDS_TRUE, *all* transactions are traced, whether or not individual tracing is specified for each transaction.
- To determine the setting of the global trace flag and to learn what types of tracing are currently enabled, use TDINFLOG.
- To determine whether tracing is turned on for a particular transaction, without listing all traced transactions, use TDINFSPT. TDINFSPT also returns the type of tracing enabled for the transaction.
- TDLSTSPT retrieves information about tracing at the mainframe server, not the TRS. Tracing at the mainframe server and at the TRS are independent of each other.
- See the Mainframe Connect Server Option *Installation and Administration Guide* for an explanation of the Gateway-Library tracing facility, instructions for using it, and the layout of the trace log.

See also

Related functions

- TDINFLOG on page 123
- TDINFSPT on page 141
- TDSETLOG on page 187
- TDSETSPT on page 206
- TDWRTLOG on page 244

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDNUMPRM

Description

Determines how many parameters were sent with the current RPC by the remote client or server.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC        PTR,
01 NUMBER_OF_PARMS FIXED BIN(31);

CALL TDNUMPRM (TDPROC, NUMBER_OF_PARMS);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

NUMBER_OF_PARMS

(O) Number of parameters accepted as part of the current RPC. This argument replaces the *RETCODE* argument for this function and is where the result of function execution is stored.

Return value

This function returns the number of parameters in the *NUMBER_OF_PARMS* argument.

Examples

The following code fragment illustrates a typical use of TDNUMPRM. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
    /*      ------------------------------------------------------------*/
    /*      accept client request                                       */
    /*      ------------------------------------------------------------*/
          CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                         SNA_CONNECTION_NAME,
                         SNA_SUBC);

/*      ------------------------------------------------------------*/
/*      TDRESULT to validate kicked off via rpc request             */
/*      ------------------------------------------------------------*/
        CALL TDRESULT (GWL_PROC, GWL_RC);
        IF GWL_RC ^= TDS_PARM_PRESENT THEN
        DO;
            CALL TDRESULT_ERROR;
            GO TO END_PROGRAM;
        END;
/*------------------------------------------------------------------*/
GET_NR_OF_PARMS:
/*------------------------------------------------------------------*/
/*      ------------------------------------------------------------*/
```

```
/*      get nr of parms .. better be two                          */
/*      ----------------------------------------------------------*/
        CALL TDNUMPRM (GWL_PROC, GWL_NUMPRM_PARMS);

        IF GWL_NUMPRM_PARMS ^= 2 THEN
        DO;
            CALL TDNUMPRM_ERROR;
            GO TO END_PROGRAM;
        END;
/*------------------------------------------------------------------*/
GET_PARMS:
/*------------------------------------------------------------------*/
```

Usage

TDNUMPRM

A server application uses this function to determine how many parameters were sent with a client RPC.

- When a cursor command is received, this function returns the number of cursor parameters for the current cursor.
- Use this function to determine how many parameters you need to retrieve with TDRCVPRM. You must call TDRCVPRM once for each parameter.

See also

Related functions

- TDACCEPT on page 68
- TDRCVPRM on page 162

TDRCVPRM

Description

Retrieves the data from an RPC parameter sent by a remote client.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC            PTR,
01 RETCODE           FIXED BIN(31),
01 PARM_ID           FIXED BIN(31),
01 HOST_VARIABLE     CHAR(n),
01 HOST_VARIABLE_TYPE FIXED BIN(31),
01 MAX_DATA_LENGTH   FIXED BIN(31),
01 ACTUAL_DATA_LENGTH FIXED BIN(31);

CALL TDRCVPRM (TDPROC, RETCODE, PARM_ID,
          HOST_VARIABLE, HOST_VARIABLE_TYPE,
           MAX_DATA_LENGTH, ACTUAL_DATA_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-24 on page 163.

PARM_ID

(I) Number of the parameter to be received. Parameters are numbered sequentially with the first parameter number one.

HOST_VARIABLE

(O) Host program variable where the parameter data is stored.

HOST_VARIABLE_TYPE

(I) Datatype of the *HOST_VARIABLE*. This is the datatype that is used in mainframe processing of this parameter.

MAX_DATA_LENGTH

(I) Maximum length of the data that can be stored in the named *HOST_VARIABLE*. For TDSVARYCHAR, TDSVARYBIN, and TDSVARYGRAPHIC parameters, this value does not include the 2 bytes for the "LL" length specification.

For graphic datatypes, this is the number of double-byte characters. For a Sybase numeric or decimal parameter, it is 35. For other datatypes, it is the number of bytes.

To determine the maximum length of the incoming data, use TDINFPRM. For fixed-length datatypes, this value is ignored.

ACTUAL_DATA_LENGTH

(O) Variable where the actual length of the received data is returned. For TDSVARYCHAR, TDSVARYBIN, and TDSVARYGRAPHIC parameters, this value does not include the 2 bytes for the "LL" length specification. If this length is greater than the specified *MAX_DATA_LENGTH*, the data is truncated, and TDRCVPRM returns TDS_ TRUNCATION_OCCURRED.

For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-24.

Table 3-24: TDRCVPRM return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_DATE_CONVERSION_ERROR (-23)	Error in conversion of datetime data. This can be a result of trying to convert short datetime (TDSDATETIME4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short datetime datatype.
TDS_DECIMAL_CONVERSION_ERROR (-24)	Error in conversion of packed decimal data.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_FLOAT_CONVERSION_ERROR (-21)	Error in conversion of float values.

Return value	Meaning
TDS_INVALID_DATA_CONVERSION (-172)	Incompatible datatypes. The source datatype cannot be converted into the requested result datatype.
TDS_INVALID_DATA_TYPE (-171)	Illegal datatype. A Sybase datatype supplied in the call is not supported and the conversion cannot be done. The operation failed.
TDS_INVALID_ID_VALUE (-10)	The specified column or parameter number is greater than the system maximum. Sybase allows as many columns per table result and parameters per RPC as the system maximum.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *xxx*-LENGTH argument is too long.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.
TDS_MONEY_CONVERSION_ERROR (-22)	Error in conversion of TDSMONEY-type data. This can be a result of trying to convert to short money (TDSMONEY4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short money datatype.
TDS_NO_PARM_PRESENT (103)	No incoming parameters present. TDRCVPRM cannot retrieve a parameter because no more parameters were accepted. The operation failed.
TDS_TRUNCATION_OCCURRED (-13)	Data was truncated. The actual data length was longer than the maximum data length allotted for this data.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

In the following code fragment, TDRCVPRM is called to receive the data in the first incoming parameter. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*-------------------------------------------------------------------*/
GET_PARMS:
/*-------------------------------------------------------------------*/
```

```
 /*      ------------------------------------------------------------*/
 /*      get return parameter information                            */
 /*      ------------------------------------------------------------*/
        GWL_INFPRM_ID = 1;
        CALL GET_PARM_INFO;

        (IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE AND
        IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE_NULLABLE) THEN
        DO;
            CALL TDINFPRM_NOT_RETURN_PARM_ERROR;
            GO TO END_PROGRAM;
        END;

        GWL_SETPRM_USER_DATA = GWL_INFPRM_USER_DATA;
        GWL_SETPRM_ID        = GWL_INFPRM_ID;
        GWL_SETPRM_DATA_L    = GWL_INFPRM_DATA_L;
        GWL_SETPRM_TYPE      = GWL_INFPRM_TYPE;

/*      ------------------------------------------------------------*/
/*      get department id parameter number from known name          */
/*      ------------------------------------------------------------*/
        GWL_INFPRM_NAME   = '@parm2';
        GWL_INFPRM_NAME_L = 6;

        CALL TDLOCPRM (GWL_PROC,
                       GWL_INFPRM_ID,
                       GWL_INFPRM_NAME,
                       GWL_INFPRM_NAME_L);

/*      ------------------------------------------------------------*/
/*      get department parameter information                        */
/*      ------------------------------------------------------------*/
        CALL GET_PARM_INFO;

        IF GWL_INFPRM_TYPE ^= TDSVARYCHAR THEN
        DO;
            CALL TDINFPRM_NOT_CHAR_PARM_ERROR;
            GO TO END_PROGRAM;
        END;
/*      ------------------------------------------------------------*/
/*      get department parameter data                               */
/*      ------------------------------------------------------------*/
        CALL TDRCVPRM (GWL_PROC, GWL_RC,
                       GWL_INFPRM_ID,
                       PARM_DEPT,
```

```
                        GWL_INFPRM_TYPE,
                        GWL_INFPRM_MAX_DATA_L,
                        GWL_RCVPRM_DATA_L);
/*--------------------------------------------------------------------*/
GET_PARM_INFO: PROC;
/*--------------------------------------------------------------------*/
        CALL TDINFPRM (GWL_PROC, GWL_RC,
                        GWL_INFPRM_ID,
                        GWL_INFPRM_TYPE,
                        GWL_INFPRM_DATA_L,
                        GWL_INFPRM_MAX_DATA_L,
                        GWL_INFPRM_STATUS,
                        GWL_INFPRM_NAME,
                        GWL_INFPRM_NAME_L,
                        GWL_INFPRM_USER_DATA);
  RETURN;

END GET_PARM_INFO;
```

Usage

TDRCVPRM

A server application calls TDRCVPRM to retrieve a parameter sent by a remote client. A server application uses TDRCVPRM only when the client request is an RPC or a cursor command. Language requests do not have parameters.

- An application must issue one TDRCVPRM call for each parameter to be retrieved. To determine the total number of parameters received, use TDNUMPRM.
- Parameters can be retrieved in any order, using the *PARM_ID* argument to specify which parameter is wanted. If you know the parameter name but not its number, call TDLOCPRM to determine the parameter ID.
- Unless you already know the length and datatype of the incoming parameter, call TDINFPRM before each TDRCVPRM call. TDINFPRM returns the datatype and length of the incoming data, and indicates whether or not it is a return parameter.
- If the *ACTUAL_DATA_LENGTH* is greater than the *MAX_DATA_LENGTH*, the data is truncated, and TDRCVPRM returns TDS_TRUNCATION_OCCURRED.
- A server program can modify the data length of a return parameter by issuing TDSETPRM before it returns results.

- If the parameter datatype is different from the one specified in *HOST_VARIABLE_TYPE*, TDRCVPRM converts it to the specified datatype before processing (*implicit conversion*).

Datatype conversions

Table 3-25 shows which implicit conversions can be performed by TDRCVPRM.

Table 3-25: Datatype conversions performed by TDRCVPRM

Source datatype: Open Client	Result datatype: Gateway-Library	Notes
TDSCHAR	TDSVARYCHAR	Performs ASCII to EBCDIC conversion. For Japanese character sets, does workstation to mainframe conversion.
TDSCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	
TDSLONGVARCHAR	TDSCHAR	Pads TDSCHAR fields with blanks.
TDSLONGVARCHAR	TDSVARYCHAR	
TDSFLT4	TDSFLT8	Truncates low order digits.
TDSFLT4	TDS_PACKED_DECIMAL	
TDSFLT8	TDSFLT4	
TDSFLT8	TDS_PACKED_DECIMAL	
TDSMONEY	TDSFLT4	
TDSMONEY	TDSFLT8	
TDSMONEY4	TDSFLT4	
TDSMONEY4	TDSFLT8	
TDSMONEY	TDSCHAR	
TDSMONEY	TDSVARYCHAR	
TDSMONEY	TDS_PACKED_DECIMAL	
TDSCHAR	TDS_PACKED_DECIMAL	
TDSVARYCHAR	TDS_PACKED_DECIMAL	
TDSNUMERIC	TDS_PACKED_DECIMAL	When receiving numeric or Sybase decimal as char, *MAX_DATA_LENGTH* is the maximum length of the result (precision +2, or precision +3 if precision=scale).
TDSNUMERIC	TDS_CHAR	
TDS_SYBASE_DECIMAL	TDS_PACKED_DECIMAL	When receiving numeric or Sybase decimal as packed decimal, use TDINFBCD to determine the host packed decimal precision and scale. *MAX_DATA_LENGTH* is the actual length of the packed decimal.
TDS_SYBASE_DECIMAL	TDS_CHAR	

Source datatype: Open Client	Result datatype: Gateway-Library	Notes
TDSCHAR	TDSGRAPHIC	Used with Japanese DBCS.
TDSCHAR TDSVARYCHAR TDSVARYCHAR	TDSVARYGRAPHIC TDSGRAPHIC TDSVARYGRAPHIC	Pads TDSGRAPHIC fields with blanks.
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	

For more information about datatypes, see "Datatypes" on page 36.

TDRCVPRM pads binary-type host variables with zeroes and graphic- or character-type host variables with blanks. No default padding is set for columns of other datatypes.

Note Open Client automatically converts all fixed character (TDSCHAR) parameters to variable character (TDSVARYCHAR) parameters when it sends them to a server. If you prefer to work with fixed character parameters, assign *HOST_VARIABLE_TYPE* a value of TDSCHAR.

For Japanese users

When the Japanese Conversion Module (JCM) is used, TDRCVPRM converts the parameter data from the client character set to the one used at the mainframe server, if conversion is necessary.

- When converting client character data to mainframe graphic data, Gateway-Library divides the length of incoming Japanese strings in half because the length of mainframe graphic datatypes is the number of double-byte characters, whereas the length of character datatypes at both the mainframe and the workstation is the number of bytes.

  Your program needs to allow for length differences between the workstation character set and the mainframe character set.

  See "Processing Japanese client requests" on page 56 and "Datatypes" on page 36 for a full discussion of character set conversions and length considerations.

- When using the JCM, an application can call TDSETSOI to manipulate Shift Out/Shift In codes for character data before issuing a TDRCVPRM call.

- Table 3-24 on page 163 lists the implicit conversions that the JCM does when retrieving data.

See also

Related functions

- TDACCEPT on page 68
- TDINFPRM on page 133
- TDLOCPRM on page 152
- TDNUMPRM on page 160
- TDRCVSQL on page 169
- TDSETPRM on page 194

Related topics

- “Character sets” on page 17
- “Datatypes” on page 36
- “Processing Japanese client requests” on page 56

TDRCVSQL

Description

Receives a language string from a remote client.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC                PTR,
01 RETCODE               FIXED BIN(31),
01 HOST_VARIABLE         CHAR(n),
01 MAX_VAR_LENGTH        FIXED BIN(31),
01 ACTUAL_STRING_LENGTH FIXED BIN(31);

CALL TDRCVSQL(TDPROC, RETCODE, HOST_VARIABLE,
              MAX_VAR_LENGTH, ACTUAL_STRING_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-26 on page 170.

HOST_VARIABLE

(O) Host program variable where the text of the retrieved language string is stored.

MAX_VAR_LENGTH

(I) Maximum length of the string that can be stored in the named *HOST_VARIABLE*. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

ACTUAL_STRING_LENGTH

(O) The actual length of the incoming data. If this length is greater than the specified *MAX_VAR_LENGTH*, the data is truncated.

Note If this is a Japanese character set, the length may be halved when converted to IBM Kanji by Gateway-Library.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-26.

Table 3-26: TDRCVSQL return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *MAX_VAR_LENGTH* argument is too short.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_NO_SQL_PRESENT (101)	No incoming language string present. TDRCVSQL cannot retrieve more text because no more text was accepted. The operation failed.
TDS_TRUNCATION_OCCURRED (-13)	Data was truncated. The actual data length was longer than the maximum data length allotted for this data.

Return value	Meaning
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates the use of TDRCVSQL to receive a SQL language request from the client. This example is taken from the sample program in Appendix C, "Sample Language Application for CICS."

```
/*-------------------------------------------------------------------*/
READ_IN_SQL_TEXT:
/*-------------------------------------------------------------------*/

/*      ---------------------------------------------------------*/
/*      prepare for receive                                      */
/*      ---------------------------------------------------------*/
       CALL TDRESULT (GWL_PROC, GWL_RC);

/*      ---------------------------------------------------------*/
/*      load ptr to redefined sql text                           */
/*      ---------------------------------------------------------*/
       LANG_BUFFER_PTR = ADDR(LANG_BUFFER_DB2);

/*      ---------------------------------------------------------*/
/*      get len of language text, ensure not too big for us      */
/*      (this could be handled without TDSQLLEN by checking      */
/*      LANG_ACTUAL_LEN doesn't exceed LANG_MAX_L in TDRCVSQL call) */
/*      ---------------------------------------------------------*/
       CALL TDSQLLEN (GWL_PROC, GWL_SQLLEN);

       LANG_MAX_L = STG(LANG_BUFFER_TEXT);

       IF GWL_SQLLEN > LANG_MAX_L THEN
       DO;
           MSG_TEXT   = MSG_BAD_LEN;
           MSG_TEXT_L = STG(MSG_BAD_LEN);
           CALL SEND_ERROR_MESSAGE;
           GO TO END_PROGRAM;
       END;

/*      ---------------------------------------------------------*/
/*      get language text                                        */
/*      ---------------------------------------------------------*/
       CALL TDRCVSQL (GWL_PROC, GWL_RC,
                      LANG_BUFFER_TEXT,
```

```
                        LANG_MAX_L,
                        LANG_ACTUAL_L);

LANG_BUFFER_LL = LANG_ACTUAL_L;
```

Usage

TDRCVSQL

A server application uses this function to retrieve a SQL or other language string from a client. Although the function is called TDRCVSQL, it can receive any type of language request, including math functions, single-byte katakana, and so on.

- TDRCVSQL does not differentiate between SQL strings and other character text strings. Your application must determine what kind of text is in the buffer and what to do with it.
- You can determine the length of the incoming string by issuing TDSQLLEN after TDACCEPT and before TDRCVSQL.
- To determine whether the incoming request is a language request, cursor request, or an RPC, call TDINFPGM or TDRESULT. In long-running transactions, TDGETREQ indicates the type of request.

  If your program calls TDRCVSQL and the request is not a language or cursor/dynamic request, TDRCVSQL returns TDS_NO_SQL_PRESENT.
- You can divide the language string between two variables. First, specify a partial length in the *MAX_VAR_LENGTH* argument of one TDRCVSQL call. Then, issue TDSQLLEN just before conversion to determine the length of the remaining text, and specify that length in the *MAX_VAR_LENGTH* argument of a subsequent TDRCVSQL call.

  Note If you are using a double-byte character set, see instructions under "For Japanese Users" to learn how to divide a string between two variables.

- If the *ACTUAL_STRING_LENGTH* of the text is longer than that specified in *MAX_VAR_LENGTH*, the string is truncated, and TDRCVSQL returns TDS_TRUNCATION_OCCURRED.

For Japanese users

To divide a language string between two variables when using double-byte character sets, set *MAX_VAR_LENGTH* to two times the length returned by TDSQLLEN.

See also

Related functions

- TDGETREQ on page 103
- TDINFPGM on page 127
- TDSQLLEN on page 232

TDRESULT

Description

Determines whether a request is pending and identify the type of object received.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC  PTR,
01 RETCODE FIXED BIN(31);

CALL TDRESULT (TDPROC, RETCODE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-27.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-27.

Table 3-27: TDRESULT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_PARM_PRESENT (203)	Parameter value received. A parameter was received from the remote client. This value is returned to TDRESULT when a parameter is accepted by a server program and is ready to be retrieved.

Return value	Meaning
TDS_RESULTS_COMPLETE (500)	TDRESULT indicated no more results. No, or no more, language text, RPC parameters, cancel requests, or messages were retrieved.
TDS_SQL_CMD_PRESENT (201)	Language string received. A language request was received from a remote client. This value is returned to TDRESULT when a language string is accepted by a server program and is ready for retrieval.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

In the following code fragment, the program calls TDRESULT to ascertain that the incoming request is an RPC, before it retrieves any parameters.
This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*      ------------------------------------------------------------*/
/*       accept client request                                      */
/*      ------------------------------------------------------------*/
       CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                      SNA_CONNECTION_NAME,
                      SNA_SUBC);
/*      ------------------------------------------------------------*/
/*       TDRESULT to validate kicked off via rpc request            */
/*      ------------------------------------------------------------*/
       CALL TDRESULT (GWL_PROC, GWL_RC);

       IF GWL_RC ^= TDS_PARM_PRESENT THEN
       DO;
           CALL TDRESULT_ERROR;
           GO TO END_PROGRAM;
       END;
```

Usage

TDRESULT

A server application can use this function to determine whether a remote client sent a new request over this connection, and, if so, what kind of request—a language request or an RPC.

- If the request is a language request, TDRESULT returns TDS_SQL_CMD_PRESENT.

- If the request is an RPC with parameters, TDRESULT returns TDS_PARM_PRESENT.
- In a long-running transaction, TDGETREQ returns the type of request pending. There is no need to call TDRESULT after TDGETREQ.
- An application can call TDRESULT to determine whether any more results are pending. After all SQL statements or RPC parameters are read in, TDRESULT returns TDS_RESULTS_COMPLETE.
- This function is not required. It is included for compatibility with earlier versions of Gateway-Library.
- Use TDINFPGM, TDGETREQ, or TDINFRPC to determine what type of request the remote client sent.

See also

Related functions

- TDACCEPT on page 68
- TDRCVPRM on page 162
- TDRCVSQL on page 169

TDSETACT

Description

Turns on or off system-wide accounting for Gateway-Library. Under CICS, rename the CICS accounting log.

Syntax

```
%INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE              FIXED BIN(31),
01 ACCOUNTING_FLAG      FIXED BIN(31),
01 ACCOUNTING_FILENAME  CHAR(08),
01 MAXNUM_TRACE_RECORDS FIXED BIN(31);

CALL TDSETACT(IHANDLE, RETCODE, ACCOUNTING_FLAG,
              ACCOUNTING_FILENAME,
              MAXNUM_ACCT_RECORDS);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-28 on page 177.

ACCOUNTING_FLAG

(I) Accounting on/off indicator. Assign this argument one of the following values:

TDS_TRUE (1)	Turn on accounting.
TDS_FALSE (0)	Turn off accounting.

ACCOUNTING_FILENAME

(I) Name of the accounting log.

Under CICS: Specify the *DATASET* name from the CICS File Control Table (FCT) entry that describes the VSAM file used for this log. As installed, this name is *SYTACCT1*. You can change the name of this log by specifying a new name here.

Under IMS TM and MVS: Leave this field blank. IMS TM and MVS ignore this value.

MAXNUM_ACCT_RECORDS

(I) Accounting log record limit.

Under CICS: This is the maximum number of records to be allocated for this accounting file. To indicate the system maximum, assign this argument a value of -1. We recommend always setting this value to -1.

Under IMS TM: The IMS TM system log does not have a limit. We recommend always using -1.

Under MVS: Use -1. The size of the log is determined by the space allocated to the sequential file used as the MVS log.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-28.

Table 3-28: TDSETACT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Return value	Meaning
TDS_LOG_ERROR (-258)	Attempt to write to the log file failed.

Examples

The following code fragment receives a request to turn accounting off, checks that accounting is on, and uses TDSETACT to turn it off. This example is based on the sample program in Appendix G, "Sample Tracing and Accounting Program," which runs under CICS.

```
/*-----------------------------------------------------------------*/
GET_PARM:
/*-----------------------------------------------------------------*/
       CALL TDRCVPRM (GWL_PROC, GWL_RC,
                      GWL_RCVPRM_ID,
                      PARM_REQUEST, TDSCHAR,
                      GWL_RCVPRM_MAX_DATA_L,
                      GWL_RCVPRM_DATA_L);

       IF PARM_REQUEST = PARM_REQUEST_SETACT_OFF THEN
           CALL TDSETACT_OFF_PROC;
/*-----------------------------------------------------------------*/
TDSETACT_OFF_PROC: PROC;
/*-----------------------------------------------------------------*/
       CALL TDINFACT (GWL_INIT_HANDLE, GWL_RC,
                      GWL_INFACT_STATUS,
                      GWL_INFACT_FILENAME,
                      GWL_INFACT_RECORDS);

       IF GWL_RC NE TDS_OK THEN
       DO;
           SEND_DONE       = SEND_DONE_ERROR;
           MSG_SRVLIB_FUNC = 'TDINFACT';
           GO TO TDSETACT_OFF_EXIT;
       END;
/*    ---------------------------------------------------------------*/
/*      Turn off host accounting, if it's on.                        */
/*    ---------------------------------------------------------------*/
       IF GWL_INFACT_STATUS = TDS_TRUE THEN
       DO;
           CALL TDSETACT (GWL_INIT_HANDLE, GWL_RC,
                          TDS_FALSE,
                          GWL_INFACT_FILENAME,
                          GWL_INFACT_RECORDS);

           IF GWL_RC NE TDS_OK THEN
           DO;
               SEND_DONE       = SEND_DONE_ERROR;
```

```
                MSG_SRVLIB_FUNC = 'TDSETACT';
                GO TO TDSETACT_OFF_EXIT;
            END;
        END;
/*-------------------------------------------------------------------*/
    TDSETACT_OFF_EXIT:
/*-------------------------------------------------------------------*/
        RETURN;

   END TDSETACT_OFF_PROC;
```

Usage

TDSETACT

You use this function to begin recording accounting information in the accounting log, to stop recording after it began, or, under CICS, to change the name of the accounting log.

- This function returns accounting information recorded at the mainframe server. The TRS administrator can turn local accounting recording on and off at the TRS. Accounting at the mainframe and at the TRS are independent of each other.
- Gateway-Library accounting records the total number of TDS bytes, packets, messages, rows, requests, and cancels sent and received at the mainframe server from the time a TDACCEPT function initializes the TDS environment until a TDFREE is issued, and the number of seconds and milliseconds that elapsed during the conversation.
- The accounting flag is set to off when Gateway-Library is initialized. It remains off until the program explicitly turns it on with TDSETACT; then it remains on until the program explicitly turns it off with TDSETACT. No other Gateway-Library functions turn accounting on or off.
- If a transaction does not call this function, the accounting flag remains in the state it was in before the transaction executed.
- TDSETACT opens the specified accounting log when it turns accounting recording on.

  Note The IMS TM system log is always open, but TDSETLOG does a logical OPEN by turning accounting on.

- Accounting information is written to the accounting log after TDFREE is issued.

- The log used for accounting depends upon the transaction processing system in use:
  - *Under CICS:* The accounting log is a VSAM ESDS file known to CICS as *SYTACCT1*.

    You can use this function to change the name of the accounting log as long as the name you specify matches an FCT *DATASET* entry. An alternate log may already exist—an FCT entry for the alternate log *SYTACCT2* is included in the installation instructions.

    When the log fills up, you must explicitly empty or delete the log or specify an alternate log in the *ACCOUNTING_FILENAME* argument.
  - *Under IMS TM:* The accounting log is the IMS TM system log. For more information on this log, see your IMS TM documentation.
  - *Under MVS:* The log file is a sequential file (usage is optional).
- See the Mainframe Connect Server Option *Installation and Administration Guide* for an explanation of the Gateway-Library accounting facility, instructions for using it, and the layout of the CICS accounting log.

See also

Related functions

- TDACCEPT on page 68
- TDFREE on page 99
- TDINFACT on page 116

Related Documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDSETBCD

Description

Sets the length and number of decimal places for a given packed decimal column or parameter. You can also set the number of decimal places for numeric and Sybase decimal columns.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC                  PTR,
01 RETCODE                 FIXED BIN(31),
01 OBJECT_TYPE             FIXED BIN(31),
01 OBJECT_NUMBER           FIXED BIN(31),
01 BCD_LENGTH              FIXED BIN(31),
01 BCD_NUMBER_DECIMAL_PLACES FIXED BIN(31);

CALL TDSETBCD(TDPROC, RETCODE, OBJECT_TYPE,
          OBJECT_NUMBER, BCD_LENGTH,
           BCD_NUMBER_DECIMAL_PLACES);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-29 on page 182.

OBJECT_TYPE

(I) Object type indicator. Indicates whether the object is a parameter or a column. Assign this argument one of the following values:

TDS_OBJECT_COL(1)	Object is a column in a return row.
TDS_OBJECT_PARM (2)	Object is a parameter.

OBJECT_NUMBER

(I) Number of the column or parameter with the information that is being set.

If the object is a column, this is the position of the column in the row, counting from left to right. Columns are numbered sequentially with the leftmost column in a row number one.

If the object is a return parameter, this is the number of the parameter with the value that is being checked. All parameters are counted, whether or not they are return parameters. Parameters are numbered sequentially with the first parameter number one.

BCD_LENGTH

(I) The length of the packed decimal field. This value must not be a negative number. The maximum allowed length for a packed decimal object is 31. Instead of a specific value, you can default to the *COLUMN_MAXLEN* specified in the TDESCRIB call that describes this column. To do this, assign this argument a value of TDS_DEFAULT_LENGTH.

BCD_NUMBER_DECIMAL_PLACES

(I) Number of decimal places in the object. This value must not be a negative number. The maximum number of decimal places allowed for a packed decimal object is 31. The maximum decimal places allowed for Sybase numeric or decimal is 77.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-29.

Table 3-29: TSETBCD return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *BCD_LENGTH* argument is wrong.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates a typical use of TDSETBCD. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.   */
/*      ------------------------------------------------------------ */
        DB_DESCRIBE_HV_PTR     = ADDR(EMPLOYEE_JC);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_JC);
        WRKLEN1                = STG(EMPLOYEE_JC);
        WRKLEN2                = STG(CN_JC);
        DB_HOST_TYPE           = TDSDECIMAL;
        DB_CLIENT_TYPE         = TDSFLT8;
        CALL DESCRIBE_COLUMN;
/*      ------------------------------------------------------------*/
```

```
/*      We must inform the Server Library how many decimal places   */
/*      are in the EMPLOYEE_JC column.                              */
/*      ------------------------------------------------------------*/
        CALL TDSETBCD (GWL_PROC, GWL_RC,
                       TDS_OBJECT_COL,
                       CTR_COLUMN,
                       TDS_DEFAULT_LENGTH,
                       GWL_SETBCD_SCALE);
/*      ------------------------------------------------------------*/
/*      Demonstrate getting decimal column information.             */
/*      ------------------------------------------------------------*/
        CALL TDINFBCD (GWL_PROC, GWL_RC,
                       TDS_OBJECT_COL,
                       CTR_COLUMN,
                       GWL_INFBCD_LENGTH,
                       GWL_INFBCD_SCALE);
```

Usage

TDSETBCD

Packed decimal data is supported in PL/1, but not in DB-Library or Client-Library. This function preserves the scale and value when converting a DB-Library or Client-Library decimal value to PL/1 packed decimal data and vice versa.

Note Although the name of this function implies BCD data, in PL/1 this function is actually used with packed decimal data.

- Always use this function when describing Sybase decimal and numeric columns, and when using TDSETPRM for implicit conversion from char or packed decimal to a Sybase numeric or decimal return parameter. Assign the following:
  - Precision to *BCD_LENGTH*
  - Scale to *BCD_NUMBER_DECIMAL_PLACES*
- Use this function to specify:
  - The length and number of decimal places of a client parameter before converting it to packed decimal.
  - The length and number of decimal places of a column that contains packed decimal data, before returning the data to the client.
  - The number of decimal places for numeric and Sybase decimal columns.

- For parameters, use this function to specify the length and number of decimal places of a TDS_MONEY type parameter before it is converted to packed decimal.

Note When reading in decimal data, call TDSETBCD before calling TDRCVPRM to set the decimal point location. When returning decimal data to a client, call TDSETBCD after calling TDESCRIB.

See also

Related functions

- TDESCRIB on page 89
- TDINFBCD on page 120

TDSETLEN

Description

Sets the column length for a variable-length field before sending it to a client.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC           PTR,
01 RETCODE          FIXED BIN(31),
01 COLUMN_NUMBER    FIXED BIN(31),
01 NEW_COLUMN_LENGTH FIXED BIN(31);

CALL TDSETLEN(TDPROC, RETCODE, COLUMN_NUMBER,
              NEW_COLUMN_LENGTH;
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-30 on page 185.

COLUMN_NUMBER

(I) The number of the column that is being described. Columns are numbered sequentially; the first column in a row is number 1.

NEW_COLUMN_LENGTH

(I) New length of the column data. This argument specifies the length of the data that is sent in subsequent TDSNDROW calls. This value must be greater than zero but cannot be greater than the maximum length of the column, as determined by TDESCRIB.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-30.

Table 3-30: TDSETLEN return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_DATA_TYPE (-171)	Illegal datatype. A Sybase datatype supplied in the call is not supported and the conversion cannot be done. The operation failed.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *NEW_COLUMN_LENGTH* argument is too long.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates a typical use of TDSETLEN. This example is taken from the sample program in Appendix B, “Sample RPC Application for CICS.”

```
/* ------------------------------------------------------------*/
/*      Do not send trailing blanks of EMPLOYEE_LNM
/* ------------------------------------------------------------*/
          WRKLEN1 = LENGTH(EMPLOYEE_LNM);
          CTR_COLUMN = 2;
          WRK_BLANKS_SS = 1;
LOOP:       DO WHILE(WRK_BLANKS_SS <= WRKLEN1);
            IF SUBSTR(EMPLOYEE_LNM, WRK_BLANKS_SS, 1) = ' ' THEN DO;
              LEAVE LOOP;
            END;
            WRK_BLANKS_SS = WRK_BLANKS_SS + 1;
          END LOOP;
          IF (WRK_BLANKS_SS <= WRKLEN1) THEN DO;
            CALL TDSETLEN (GWL_PROC, GWL_RC,
                           CTR_COLUMN,
```

```
                                WRK_BLANKS_SS - 1);
END;
```

Usage

TDSETLEN

A server application uses this function to specify the length of data for a single column, before it is sent to the client.

- Column data lengths are initially set with TDESCRIB. For fixed-length fields, there is no need to set the column lengths again. For variable-length fields, if the actual data length changes from one row to another, your application needs to reset the column length before you send the row of data to the client.
- Your application must issue a separate TDSETLEN for each column for which the data length changes.
- Each column of the row must first be defined in a TDESCRIB statement. The TDSETLEN statement must be coded after the TDESCRIB statement for that column.
- Your application must be in SEND state for this function to execute successfully. If it is not in SEND state, TDSETLEN returns TDS_WRONG_STATE. To switch to SEND state, call TDRESULT.
- The column length set by TDSETLEN must not be greater than the maximum column length specified in TDESCRIB. If it is longer, the function returns TDS_INVALID_LENGTH.

See also

Related functions

- TDESCRIB on page 89
- TDSNDROW on page 227

TDSETLOG

Description

Sets system-wide tracing for the mainframe server and rename the CICS trace log.

Syntax

```
% INCLUDE SYGWPLI;

01 IHANDLE              PTR,
01 RETCODE               FIXED BIN(31),
01 GLOBAL_TRACE_FLAG     FIXED BIN(31),
01 API_TRACE_FLAG        FIXED BIN(31),
01 TDS_HEADER_TRACE_FLAG FIXED BIN(31),
01 TDS_DATA_TRACE_FLAG   FIXED BIN(31),
01 TRACE_ID              FIXED BIN(31),
01 TRACE_FILENAME        CHAR(08) INIT('  '),
01 MAXNUM_TRACE_RECORDS  FIXED BIN(31);

CALL TDSETLOG(IHANDLE, RETCODE, GLOBAL_TRACE_FLAG,
              API_TRACE_FLAG, TDS_HEADER_TRACE_FLAG,
              TDS_DATA_TRACE_FLAG, TRACE_ID,
               TRACE_FILENAME, MAXNUM_TRACE_RECORDS);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-31 on page 189.

GLOBAL_TRACE_FLAG

(I) Global or specific trace indicator. Use this argument to turn tracing on or off and to indicate whether tracing is global (trace all transactions) or applies to a specific set of transactions. If tracing is off, only errors are logged.

Specific tracing can be set for 1 through 8 transactions. To set tracing for a particular transaction, use TDSETSPT.

Assign this argument one of the following values:

TDS_NO_TRACING (0)	Turn off all tracing.
TDS_TRACE_ALL_RPCS (1)	Turn on global tracing.s
TDS_TRACE_SPECIFIC_RPCS (2)	Turn on specific tracing.
TDS_TRACE_ERRORS_ONLY (3)	Log errors only.

API_TRACE_FLAG

(I) API tracing on/off indicator. This is a Boolean value that sets tracing on or off for Gateway-Library calls. Assign this argument one of the following values:

TDS_TRUE (1)	Turn on API tracing.
TDS_FALSE (0)	Turn off API tracing.

TDS_HEADER_TRACE_FLAG

(I) TDS header tracing on/off indicator. This is a Boolean value that sets tracing on or off for TDS headers. Assign this argument one of the following values:

TDS_TRUE (1)	Turn on header tracing.
TDS_FALSE (0)	Turn off header tracing.

TDS_DATA_TRACE_FLAG

(I) TDS data tracing on/off indicator. This is a Boolean value that sets tracing on or off for TDS data. Assign this argument one of the following values:

TDS_TRUE (1)	Turn on data tracing.
TDS_FALSE (0)	Turn off data tracing.

TRACE_ID

(I) The trace entry identifier.

Under CICS: This is the tag for the auxiliary file entry.

Under IMS TM and MVS: Leave this field blank. This argument is ignored.

TRACE_FILENAME

(I) Name of the trace/error log.

Under CICS: Specify the *DATASET* name from the CICS File Control Table (FCT) entry that describes the VSAM file used for this log. As installed, this name is *SYTDLOG1*. You can change the name of this log by specifying a new name here.

Under IMS TM and MVS: Leave this field blank. IMS TM and MVS ignore this value.

MAXNUM_TRACE_RECORDS

(I) Trace log record limit.

Under CICS: This is the maximum number of records to be allocated for this trace file. To indicate the system maximum, assign this argument a value of -1. We recommend always using -1.

Under IMS TM: The IMS TM system log does not have a limit. We recommend always using -1.

Under MVS: Use -1. The size of the log is determined by the space allocated to the sequential file used as the MVS log.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-31 on page 189.

Table 3-31: TDSETLOG return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_LOG_ERROR(-258)	Attempt to write to the log file failed.

Examples

Example 1

The following example illustrates the use of TDSETLOG to enable API tracing and header tracing. It is taken from the sample program in Appendix G, "Sample Tracing and Accounting Program," which runs under CICS.

```
/*-------------------------------------------------------------------*/
      TDSETLOG_ON_PROC: PROC;
/*-------------------------------------------------------------------*/
       CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                      GWL_INFLOG_GLOBAL,
                      GWL_INFLOG_API,
                      GWL_INFLOG_HEADER,
                      GWL_INFLOG_DATA,
                      GWL_INFLOG_TRACEID,
                      GWL_INFLOG_FILENAME,
                      GWL_INFLOG_RECORDS);

       IF GWL_RC NE TDS_OK THEN
       DO;
           SEND_DONE       = SEND_DONE_ERROR;
```

```
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDSETLOG_ON_EXIT;
        END;
/*      ------------------------------------------------------------*/
/*      turn on API (CICS Aux Trace) and Header tracing             */
/*      ------------------------------------------------------------*/
        CALL TDSETLOG (GWL_INIT_HANDLE, GWL_RC,
                       TDS_TRACE_ALL_RPCS,
                       TDS_TRUE,
                       TDS_TRUE,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETLOG';
            GO TO TDSETLOG_ON_EXIT;
        END;

/*--------------------------------------------------------------------*/
TDSETLOG_ON_EXIT:
/*--------------------------------------------------------------------*/
        RETURN;

END TDSETLOG_ON_PROC;
```

Example 2

The following example illustrates the use of TDSETLOG to disable API tracing and header tracing. It is taken from the sample program in Appendix G, "Sample Tracing and Accounting Program," which runs under CICS.

```
 /*--------------------------------------------------------------------*/
 TDSETLOG_OFF_PROC: PROC;
 /*--------------------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);
```

```
            IF GWL_RC NE TDS_OK THEN
            DO;
                SEND_DONE       = SEND_DONE_ERROR;
                MSG_SRVLIB_FUNC = 'TDINFLOG';
                GO TO TDSETLOG_OFF_EXIT;
            END;
        /*      ------------------------------------------------------------*/
        /*      turn off API (CICS Aux Trace) and Header tracing            */
        /*      ------------------------------------------------------------*/
            CALL TDSETLOG (GWL_INIT_HANDLE, GWL_RC,
                           TDS_NO_TRACING,
                           TDS_FALSE,
                           TDS_FALSE,
                           GWL_INFLOG_DATA,
                           GWL_INFLOG_TRACEID,
                           GWL_INFLOG_FILENAME,
                           GWL_INFLOG_RECORDS);

            IF GWL_RC NE TDS_OK THEN
            DO;
                SEND_DONE       = SEND_DONE_ERROR;
                MSG_SRVLIB_FUNC = 'TDSETLOG';
                GO TO TDSETLOG_OFF_EXIT;
            END;
        /*--------------------------------------------------------------------*/
        TDSETLOG_OFF_EXIT:
        /*--------------------------------------------------------------------*/
            RETURN;

        END TDSETLOG_OFF_PROC;
```

Usage

TDSETLOG

You use this function to turn on or off one or more kinds of tracing, and to specify whether tracing is global or for specific transactions only.
The following kinds of tracing are supported:

- API call tracing. Traces Gateway-Library calls under the following communication font-ends:
  - *Under CICS*. The API tracing uses the CICS auxiliary trace facility.
  - *Under IMS TM*. The API tracing uses the IMS TM system log.
  - *Under MVS*. MVS uses a sequential file.
- TDS header tracing. Keeps track of the 8-byte TDS headers that are sent to and from the mainframe server.

- TDS data tracing. Traces both incoming and outgoing TDS data.

Note The trace log is also the error log.

Trace flag

- The trace flag is set to off when the Gateway-Library is initialized. It remains off until the program explicitly turns it on with TDSETLOG, and remains on until the program explicitly turns it off with TDSETLOG. No other Gateway-Library functions turn tracing on or off.
- Specific tracing can be set for up to eight transactions. To set tracing for a particular transaction, use TDSETSPT. To find out whether specific tracing is set for a particular transaction, call TDINFSPT. For a list of the transactions being specifically traced, call TDLSTSPT.

Global trace flag

- The specified types of tracing described previously apply to all transactions if the *GLOBAL-TRACE-FLAG* is set to TDS-TRACE-ALL-RPCS. If the *GLOBAL-TRACE-FLAG* is set to TDS-TRACE-SPECIFIC-RPCS, tracing applies to only those transactions specified in TDSETSPT calls.
- If the global trace flag is set to TDS-NO-TRACING or TDS-TRAC-ERRORS, the program ignores the settings for the API, TDS header, and TDS data flags and turns them off.
- A transaction can call this function any time after TDINIT. To set tracing on for the entire transaction, code this function before TDACCEPT. To set tracing on for only a portion of a transaction, use TDSETLOG anywhere within your program.
- TDSETLOG begins writing to the specified log when it turns tracing on. It appends each new trace or error record to the trace log.
- If your program does not call this function, the trace flag remains in the state it was in before the transaction executed.

Trace log

- The log used for tracing depends upon the transaction processing system in use and the type of tracing that is done.
  - *Under CICS:* Header and data traces are written to the trace log. The trace log is a VSAM ESDS file.

As installed, the CICS trace log is named *SYTDLOG1*. You can change the name of this file by specifying a different name in the *TRACE_FILENAME* argument. The new name must match an FCT *DATASET* entry. Note that an alternate log may already exist. An FCT entry for the alternate log *SYTDLOG2* is included in the installation instructions.

When the VSAM log fills up, you must explicitly empty or delete the log or specify an alternate log in the *TRACE_FILENAME* argument.

API tracing uses the CICS auxiliary facility. CICS users can retrieve the auxiliary trace output with the CICS job stream or with third party packages designed for this purpose. Refer to your CICS documentation for details about this facility.

- *Under IMS TM:* Header, data, and API tracing information are all written to the IMS TM system log. The same log is used for errors, tracing, and accounting, so each record needs to indicate which type of record it is.

  The layout of this log is the same as the layout of the CICS log except for the header. For details about the IMS TM system log, refer to your IMS TM documentation.

- *Under MVS:* The layout of this log is the same as the layout of the CICS log.

Tracing at the mainframe server

- This function governs tracing at the mainframe server. The TRS administrator can turn tracing on and off at the TRS. Tracing at the mainframe server and at the TRS are independent of each other.

See also

Related functions

- TDACCEPT on page 68
- TDFREE on page 99
- TDINFLOG on page 123
- TDINFSPT on page 141
- TDSETSPT on page 206
- TDWRTLOG on page 244

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDSETPRM

Description

Specifies the content and length of a return parameter before returning it to a remote client.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC              PTR,
01 RETCODE             FIXED BIN(31),
01 PARM_ID             FIXED BIN(31),
01 HOST_VARIABLE_TYPE   FIXED BIN(31),
01 HOST_VARIABLE_LENGTH FIXED BIN(31),
01 HOST_VARIABLE        CHAR(n),
01 USER_DATATYPE        FIXED BIN(31);

CALL TDSETPRM(TDPROC, RETCODE, PARM_ID,
          HOST_VARIABLE_TYPE,
          HOST_VARIABLE_LENGTH,
          HOST_VARIABLE, USER_DATATYPE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-32 on page 195.

PARM_ID

(I) Number of the parameter to be returned. This must be the same parameter ID specified in the TDRCVPRM call that retrieved this parameter. Parameters are numbered sequentially in the order received, from 1 to 255.

HOST_VARIABLE_TYPE

(I) Datatype of the *HOST_VARIABLE*.

HOST_VARIABLE_LENGTH

(I) Length of the *HOST_VARIABLE*.

If *HOST_VARIABLE_TYPE* is TDSVARYCHAR, TDSVARYBIN, or TDSVARYGRAPHIC, this length does not include the 2 bytes for the “LL” length specification. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes (actual length).

HOST_VARIABLE

(I) Name of the host program variable that contains the return data.

USER_DATATYPE

(I) The client-specified datatype of the parameter, if any. If no user datatype is specified, code 0 for this field. Currently, this argument is ignored.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-32.

Table 3-32: TDSETPRM return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_DATE_CONVERSION_ERROR (-23)	Error in conversion of datetime data. This can be a result of trying to convert short datetime (TDSDATETIME4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short datetime datatype.
TDS_DECIMAL_CONVERSION_ERROR (-24)	Error in conversion of packed decimal data.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_FLOAT_CONVERSION_ERROR (-21)	Error in conversion of float values.
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).
TDS_INVALID_DATA_CONVERSION (-172)	Incompatible datatypes. The source datatype cannot be converted into the requested result datatype.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *HOST_VARIABLE_LENGTH* argument is too long.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.
TDS_MONEY_CONVERSION_ERROR (-22)	Error in conversion of TDSMONEY-type data. This can be a result of trying to convert to short money (TDSMONEY4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short money datatype.
TDS_TRUNCATION_ERROR (-20)	Error occurred in truncation of data value.

Return value	Meaning
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates a typical use of TDSETPRM. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*-------------------------------------------------------------------*/
GET_PARMS:
/*-------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      get return parameter information                            */
/*      ------------------------------------------------------------*/
       GWL_INFPRM_ID = 1;
       CALL GET_PARM_INFO;

       (IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE AND
       IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE_NULLABLE) THEN
       DO;
           CALL TDINFPRM_NOT_RETURN_PARM_ERROR;
           GO TO END_PROGRAM;
       END;

       GWL_SETPRM_USER_DATA = GWL_INFPRM_USER_DATA;
       GWL_SETPRM_ID        = GWL_INFPRM_ID;
       GWL_SETPRM_DATA_L    = GWL_INFPRM_DATA_L;
       GWL_SETPRM_TYPE      = GWL_INFPRM_TYPE;

/*      ------------------------------------------------------------*/
/*      get department id parameter number from known name          */
/*      ------------------------------------------------------------*/
       GWL_INFPRM_NAME   = '@parm2';
       GWL_INFPRM_NAME_L = 6;

       CALL TDLOCPRM (GWL_PROC,
                      GWL_INFPRM_ID,
                      GWL_INFPRM_NAME,
                      GWL_INFPRM_NAME_L);
```

```
/*      ------------------------------------------------------------*/
/*      get department parameter information                        */
/*      ------------------------------------------------------------*/
        CALL GET_PARM_INFO;

        IF GWL_INFPRM_TYPE ^= TDSVARYCHAR THEN
        DO;
            CALL TDINFPRM_NOT_CHAR_PARM_ERROR;
            GO TO END_PROGRAM;
        END;



/*      ------------------------------------------------------------*/
/*      get department parameter data                               */
/*      ------------------------------------------------------------*/
        CALL TDRCVPRM (GWL_PROC, GWL_RC,
                       GWL_INFPRM_ID,
                       PARM_DEPT,
                       GWL_INFPRM_TYPE,
                       GWL_INFPRM_MAX_DATA_L,
                       GWL_RCVPRM_DATA_L);

/*--------------------------------------------------------------------*/
END_OF_QUERY:
/*--------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      close cursor                                                */
/*      ------------------------------------------------------------*/
        EXEC SQL CLOSE ECURSOR;

/*      ------------------------------------------------------------*/
/*      update return parameter with nr of rows fetched             */
/*      ------------------------------------------------------------*/
        CALL TDSETPRM (GWL_PROC, GWL_RC,
                       GWL_SETPRM_ID,
                       GWL_SETPRM_TYPE,
                       GWL_SETPRM_DATA_L,

                       PARM_RETURN_ROWS,
                       GWL_SETPRM_USER_DATA);

        GO TO END_PROGRAM;
```

Usage

TDSETPRM

A server application uses this function to tell TDSNDDON where to find the data for a return parameter, as well as the data length and datatype.

- TDSETPRM sets the return value for a parameter but does not actually send it to the client. All return parameters, whether their return values were changed by TDSETPRM or not, are sent to the client when TDSNDDON is called.
- TDSETPRM is the only way to change the content or the length of a return parameter. When you call TDSNDDON, any return parameters with values that were not changed by a TDSETPRM call contain the same data they contained when received from the client.
- A return parameter must be identified as such in the *PARM_STATUS* argument of TDINFPRM. If you try to change the return value for a parameter that was not invoked as a return parameter, an error occurs.

  A valid return parameter has the *PARM_STATUS* field set to TDS_RETURN_VALUE (X'01') or TDS_RETURN_VALUE_NULLABLE(X'33') and a parameter ID of 1 or greater. Any other *PARM_STATUS* and a *PARM_ID* of 0 indicate that the parameter is not a return parameter.
- A server program may specify its own datatype for a parameter. To specify the datatype for the return value, assign it to the *USER_DATATYPE* argument.
- If the variable datatype is TDSVARYCHAR, TDSVARYBIN, or TDSVARYGRAPHIC, the length does not include the 2 bytes for the “LL” specification. The length specified in “LL” is ignored unless a -1 is coded as the length argument, in which case the length specified in the “LL” is used.
- When converting from char or packed decimal datatype to the client numeric or Sybase decimal datatype, use TDSETBCD before TDSETPRM to set precision and scale of the client datatype.

Datatype conversions

When sending data to a client, TDSETPRM converts many datatypes from the Gateway-Library (source) datatype to the client (result) datatype. Table 3-33 on page 199 shows what conversions are possible.

Table 3-33: Datatype conversions performed by TDSETPRM

Source datatype: Gateway-Library	Result datatype: Open Client	Notes
TDSCHAR	TDSVARYCHAR	Does EBCDIC to ASCII conversion. For Japanese characters, converts to workstation datatype.
TDSCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	
TDSLONGVARCHAR	TDSCHAR	
TDSLONGVARCHAR	TDSVARYCHAR	Pads TDSCHAR fields with blanks.
TDSMONEY	TDSCHAR	
TDSMONEY	TDSVARYCHAR	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	
TDSFLT4	TDSFLT8	
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSCHAR	TDSMONEY	
TDSVARYCHAR	TDSMONEY	
TDS_PACKED_DECIMAL	TDSCHAR	When converting packed decimal to character values, change the length to allow for unpacking, leading or trailing zeros, the sign and the decimal point.
TDS_PACKED_DECIMAL	TDSVARYCHAR	
TDS_PACKED_DECIMAL	TDSFLT8	
TDS_PACKED_DECIMAL	TDSMONEY	
TDSGRAPHIC	TDSCHAR	Used with Japanese double-byte character sets. Pads TDSCHAR fields with blanks.
TDSGRAPHIC	TDSVARYCHAR	
TDSVARYGRAPHIC	TDSCHAR	
TDSVARYGRAPHIC	TDSVARYCHAR	
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	
TDSCHAR	TDSNUMERIC	Use TDSETBCD to set Sybase numeric and decimal precision and scale before TDSETPRM.
TDSCHAR	TDS_SYBASE_DECIMAL	
TDS_PACKED_DECIMAL	TDSNUMERIC	Use TDSETBCD to set Sybase numeric and decimal precision and scale before TDSETPRM.
TDS_PACKED_DECIMAL	TDS_SYBASE_DECIMAL	

See also

Related functions

- TDINFPRM on page 133
- TDRCVPRM on page 162
- TDSNDDON on page 212

TDSETPT

Description

Specifies the type of IMS TM transaction being used.

Note This function is for use with IMS TM programs only. CICS programs ignore this call, however, MVS programs do not.

Syntax

```
% INCLUDE SYGWPLI;

01 IHANDLE   PTR,
01 RETCODE   FIXED BIN(31),
01 PROG_TYPE CHAR(4),
01 SPA       CHAR(n),
01 RESERVED1 FIXED BIN(31),
01 RESERVED2 FIXED BIN(31);

CALL TDSETPT (IHANDLE, RETCODE, PROG_TYPE, SPA,
          RESERVED1,RESERVED2);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-34 on page 201.

PROG_TYPE

(I) Type of IMS TM program being called. This is a four-byte padded field.

Assign this argument one of the following IMS TM program types:

MPP	An IMS TM online (implicit or Adapter) message processing program that runs in an IMS TM message processing region. This is the default.
BMP	An IMS TM batch message program that runs in an IMS TM batch message processing region.
CONV	An IMS TM message processing program that uses the IMS TM scratch pad area (SPA).
EXPL	An IMS TM message processing program that uses the explicit API. This is the only option that supports long-running transactions.

Under CICS: If you leave this field blank, Gateway-Library ignores this value and assumes a standard CICS program.

Under IMS TM: If you leave this field blank, Gateway-Library assumes a standard IMS TM MPP program.

Under MVS: *PROG_TYPE* must be EXPL.

SPA

(I) The IMS TM scratch pad area where conversational transaction results are stored.

When *PROG_TYPE* is CONV, this argument is required. For other program types, set this field to zeroes, and Gateway-Library ignores this field.

RESERVED1

(I) Reserved for future use.

RESERVED2

(I) Reserved for future use.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-34.

Table 3-34: TDSETPT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Examples

The following code fragment illustrates the use of TDINIT, TDSETPT, and TDACCEPT at the beginning of a Gateway-Library program. This example is taken from the sample program SYIXSAM1, listed in Appendix D, "Sample RPC Application for IMS TM (Explicit)."

```
/* ------------------------------------------------------------*/
/* establish gateway environment                               */
/* ------------------------------------------------------------*/
 CALL TDINIT (P_PCBTERM, GWL_RC, GWL_INIT_HANDLE);
            .
            .   [check return code]
            .
/* ------------------------------------------------------------*/
/* set program type to MPP */
/* ------------------------------------------------------------*/
 GWL_PROG_TYPE = 'EXPL';
 CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
 GWL_SPA_PTR, TDS_NULL, TDS_NULL);
            .
            .   [check return code]
            .
/* ------------------------------------------------------------*/
/* accept client request */
/* ------------------------------------------------------------*/
 CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
 SNA_CONNECTION_NAME,
 SNA_SUBC);
            .   [check return code]
```

Usage

TDSETPT

This function tells Gateway-Library which type of IMS TM transaction is being called and, if the transaction is conversational (CONV), the address of the scratch pad area.

- TDSETPT is used with IMS TM programs only. If this function is called in a CICS program, Gateway-Library ignores the function and assumes that the program is a standard CICS program.
- TDSETPT follows TDINIT and precedes TDACCEPT in a Gateway-Library program.

Because the default program type is MPP, coding TDSETPT immediately after TDINIT is particularly important for BMP, conversational (CONV), and explicit (EXPL) programs.

Note If your transaction is conversational (CONV), you must insert the scratch pad area into the IO/PCB before sending the results with TDSNDROW.

See also

Related functions

- TDACCEPT on page 68
- TDGETREQ on page 103
- TDINIT on page 148

Related topics

- "Long-running transactions" on page 52

Related documentation

- Mainframe Connect Server Option for IMS and MVS *Installation and Administration Guide*

TDSETSOI

Description

Sets the Shift Out/Shift In ("SO/SI") processing options for a column or parameter.

Note This function is used with the Japanese Conversion Module (JCM).

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC       PTR,
01 RETCODE      FIXED BIN(31),
01 OBJECT_TYPE  FIXED BIN(31),
01 OBJECT_NUMBER FIXED BIN(31),
01 STRIP_SOSI   FIXED BIN(31);

CALL TDSETSOI (TDPROC, RETCODE, OBJECT_TYPE,
          OBJECT_NUMBER, STRIP_SOSI);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-35 on page 205.

OBJECT_TYPE

(I) Indicator for type of object being set. This argument indicates whether the object being described is a column in a return row or a return parameter.

Assign this argument one of the following values:

TDS_OBJECT_COL (1)	Object is a column in a return row.
TDS_OBJECT_PARM (2)	Object is a return parameter.

OBJECT_NUMBER

(I) Number of the column or parameter being set.

If the object is a column, this is the number of the column with the SO/SI option that is being set. Columns are numbered sequentially; the first column in a row is number 1.

If the object is a parameter, this is the number of the parameter with the SO/SI option that is being set. All parameters are counted, whether or not they are return parameters. Parameters are numbered sequentially; the first parameter is number 1.

STRIP_SOSI

(I) The SO/SI processing option being set for this column or parameter.

Assign *STRIP_SOSI* one of the following values:

TDS_STRIP_SOSI (0)	SO/SI codes are stripped at the mainframe before being sent to the client. This is the default.
TDS_BLANK_SOSI (1)	SO/SI codes are converted to blanks before being sent to the client. The length of the object does not change.

Return value

The *RETCODE* argument can contain any of the return values in Table 3-35.

Table 3-35: TDSETSOI return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_FLAGS (-176)	Invalid padding option for a field.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Usage

TDSETSOI

Use this function to specify whether SO/SI codes are stripped or converted to blanks for a specified column or parameter before results are returned to the client.

- SO/SI codes are inserted around double-byte character strings when the client request is received by the Gateway-Library program. The TDSETSOI setting determines what happens to those codes when the string is returned to the client.
- If a program uses TDSETSOI to handle SO/SI codes when there are no SO/SI codes or blanks surrounding kanji characters, the TDSETSOI setting is ignored.
- SO/SI codes are used with character datatypes. Graphic datatypes do not use SO/SI codes.
- Replacing SO/SI codes with blanks maintains the length of the string. Otherwise, if SO/SI codes are stripped, the result length is shorter than the source length. Unless you know in advance how many pairs of SO/SI codes are in the source string, it is difficult to know what the result length will be.
- For a discussion of Shift Out and Shift In codes, read "Character sets" on page 17 and "Processing Japanese client requests" on page 56.

See also

Related functions

- TDGETSOI on page 110

Related topics

- "Character sets" on page 17

TDSETSPT

Description

Sets tracing on or off for a specified transaction.

Syntax

```
% INCLUDE SYGWPLI;

01 IHANDLE             PTR,
01 RETCODE             FIXED BIN(31),
01 TRACE_STATUS        FIXED BIN(31),
01 TRACE_OPTIONS       FIXED BIN(31),
01 TRANSACTION_ID      CHAR(n),
01 TRANSACTION_ID_LENGTH FIXED BIN(31);

CALL TDSETSPT(IHANDLE, RETCODE, TRACE_STATUS,
          TRACE_OPTIONS, TRANSACTION_ID,
          TRANSACTION_ID_LENGTH);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-36 on page 207.

TRACE_STATUS

(I) Trace indicator for the specified transaction. This is a Boolean value that turns tracing on or off for the specified transaction.

Assign this argument one of the following values:

TDS_TRUE (1)	Turn on tracing for this transaction.
TDS_FALSE (0)	Turn off tracing for this transaction.

TRACE_OPTIONS

(I) Type of tracing to be enabled for the specified transaction.

Assign this argument one of the following values:

TDS_SPT_API_TRACE (0x08)	Trace all Gateway-Library calls.
TDS_SPT_ERRLOG (0x02)	Enable error log recording.
TDS_SPT_TDS_DATA (0x01)	Enable TDS packet-tracing recording.

TRANSACTION_ID

(I) Mainframe transaction identifier of the affected transaction.

Under CICS: This is the *TRANSID* from the CICS Program Control Table (PCT).

Under IMS TM: This is the transaction name defined when the system is generated.

Under MVS: This is the APPC transaction name defined in the transaction profile.

TRANSACTION_ID_LENGTH

(I) Length of the *TRANSACTION_ID*.

For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes. This value is returned by TDINFSPT.

Under CICS: For CICS Version 1.7, This value is always 4 or less. For later versions, it is the actual length of the transaction ID, which can be greater than 4.

Under IMS TM: This value is always 8 or less.

Under MVS: This is the APPC transaction name defined in the transaction profile. This value is normally 8 or less.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-36.

Table 3-36: TDSETSPT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_DUPLICATE_ENTRY (-9)	Duplicate column description. You attempted to describe the same column twice with a TDESCRIB statement. The operation failed.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_STATUS (-174)	Invalid status value. The value entered in the STATUS field is invalid.

Return value	Meaning
TDS_SOS (-257)	Memory shortage. The host subsystem was unable to allocate enough memory for the control block that Gateway-Library tried to create. The operation failed.

Examples

The following code fragment illustrates the use of TDSETSPT to turn on specific tracing at the beginning of an application. This example is taken from the sample program in Appendix C, "Sample Language Application for CICS."

```
/*      ------------------------------------------------------------*/
/*      establish gateway environment                               */
/*      ------------------------------------------------------------*/
       CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);

/*      ------------------------------------------------------------*/
/*      turn on local tracing if not on globally or locally         */
/*      ------------------------------------------------------------*/
       CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                      GWL_INFLOG_GLOBAL,
                      GWL_INFLOG_API,
                      GWL_INFLOG_TDS_HEADER,
                      GWL_INFLOG_TDS_DATA,
                      GWL_INFLOG_TRACE_ID,
                      GWL_INFLOG_FILENAME,
                      GWL_INFLOG_TOTAL_RECS);

       IF GWL_INFLOG_GLOBAL ^= TDS_TRACE_ALL_RPCS &
          GWL_INFLOG_GLOBAL ^= TDS_TRACE_SPECIFIC_RPCS THEN
       DO;
           TRACING_SET_SW = TRACING_SET;
           CALL LOCAL_TRACING;
       END;

/*--------------------------------------------------------------------*/
LOCAL_TRACING: PROC;
/*--------------------------------------------------------------------*/
       CALL TDSETSPT (GWL_INIT_HANDLE, GWL_RC,
                      TRACING_SET_SW,
                      GWL_SETSPT_TRACE_LEVEL,
                      GWL_SETSPT_RPC_NAME,
                      GWL_SETSPT_RPC_NAME_L);

       RETURN;
END LOCAL_TRACING;
```

Usage

TDSETSPT

This function turns tracing on or off for the specified transaction.

- Transaction-level tracing occurs when TDSETLOG sets the global trace flag to TDS_TRACE_SPECIFIC_RPCS and sets one or more types of tracing (for example, API tracing or header tracing). Use TDINFLOG to determine the setting of the global trace flag and to learn which types of tracing are currently enabled. Call TDSETLOG to change those settings.
- If you request tracing for a transaction, and tracing is already on for that transaction, TDSETSPT returns TDS_DUPLICATE_ENTRY.
- You can turn on transaction-level tracing for up to eight (8) transactions at a time.
- Since eight is the maximum number of transactions for which tracing can be enabled at one time, you must turn tracing off for one of these transactions before you can enable tracing for an additional transaction. If you request tracing for a transaction, and eight transactions already have tracing turned on, TDSETSPT returns TDS_SOS.
- If you try to turn tracing off for a transaction for which tracing is not enabled, TDSETSPT returns TDS_ENTRY_NOT_FOUND.
- This function governs tracing at the mainframe server. The TRS administrator can turn tracing on and off at the TRS. Tracing at the mainframe server and at the TRS are independent of each other.

See also

Related functions

- TDINFLOG on page 123
- TDINFSPT on page 141
- TDLSTSPT on page 155
- TDSETLOG on page 187
- TDWRTLOG on page 244

Related Documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDSETUDT

Description

Sets the user-defined datatype for the specified column.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC       PTR,
01 RETCODE      FIXED BIN(31),
01 COLUMN_NUMBER FIXED BIN(31),
01 USER_DATATYPE FIXED BIN(31);

CALL TDSETUDT(TDPROC, RETCODE, COLUMN_NUMBER,
              USER_DATATYPE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-37.

COLUMN_NUMBER

(I) Number of the column with the datatype that is being set.

USER_DATATYPE

(I) The user-defined datatype to be assigned to the specified column.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-37.

Table 3-37: TDSETUDT return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ENTRY_NOT_FOUND (-8)	The specified column number, transaction number, or parameter does not exist.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.

Examples

The following code fragment illustrates a typical use of TDSETUDT. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*-------------------------------------------------------------------*/
SETUP_REPLY_COLUMNS:
/*-------------------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_FNM);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_FNM);
      WRKLEN1               = STG(EMPLOYEE_FNM)-2;
      WRKLEN2               = STG(CN_FNM);
      DB_HOST_TYPE          = TDSVARYCHAR;
      DB_CLIENT_TYPE        = TDSVARYCHAR;
      CALL DESCRIBE_COLUMN;

/*      ---------------------------------------------------------*/
/*      Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR) */
/*      to DBCHAR.                                                 */
/*      ---------------------------------------------------------*/
      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_LNM);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_LNM);
      WRKLEN1               = STG(EMPLOYEE_LNM)-2;
      WRKLEN2               = STG(CN_LNM);
      DB_HOST_TYPE          = TDSVARYCHAR;
      DB_CLIENT_TYPE        = TDSCHAR;
      CALL DESCRIBE_COLUMN;
      DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_ED);
      DB_COLUMN_NAME_HV_PTR = ADDR(CN_ED);
      WRKLEN1               = STG(EMPLOYEE_ED);
      WRKLEN2               = STG(CN_ED);
      DB_HOST_TYPE          = TDSINT2;
      DB_CLIENT_TYPE        = TDSINT2;
      CALL DESCRIBE_COLUMN;

/*      ---------------------------------------------------------*/
/*      Get the user defined datatype of EMPLOYEE_ED column.       */
/*      ---------------------------------------------------------*/
      CALL TDINFUDT (GWL_PROC, GWL_RC,
                     CTR_COLUMN,
                     GWL_INFUDT_USER_TYPE);
/*      ---------------------------------------------------------*/
/*      Set the user defined datatype of EMPLOYEE_ED column.       */
/*      ---------------------------------------------------------*/
      CALL TDSETUDT (GWL_PROC, GWL_RC,
                     CTR_COLUMN,
                     GWL_INFUDT_USER_TYPE);
```

Usage

TDSETUDT

Use this function to associate the user-defined datatype with a column when you return that column to the client.

- Use TDINFUDT to find out what datatype the client assigned to a given column.
- The Gateway-Library datatype for a column is specified by TDESCRIB.
- You can query and set the user-defined datatype for a return parameter with TDINFPRM and TDSETPRM.

See also

Related functions

- TDINFPRM on page 133
- TDSETPRM on page 194
- TDSETUDT on page 210

TDSNDDON

Description

Sends a results completion indication to the client.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC              PTR,
01 RETCODE             FIXED BIN(31),
01 STATUS              FIXED BIN(31),
01 ROW_COUNT           FIXED BIN(31),
01 RETURN_STATUS_NUMBER FIXED BIN(31),
01 CONN_OPTIONS        FIXED BIN(31);

CALL TDSNDDON(TDPROC, RETCODE, STATUS,ROW_COUNT,
          RETURN_STATUS_NUMBER, CONN_OPTIONS);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-39 on page 215.

STATUS

(I) The result of the operation. Assign this argument one of the following values:

TDS_DONE_FINAL (0x0000)	The set of results currently being sent is the final set of results. If *STATUS* is TDS_DONE_FINAL, *CONN_OPTIONS* must be TDS_ENDREPLY or TDS_ENDRPC. *Note:* TDS_ENDREPLY is not supported for the IMS TM implicit API.
TDS_DONE_CONTINUE (0x0001)	More results follow. This option tells the receiving program to continue retrieving results until this argument specifies TDS_DONE_FINAL or TDS_DONE_ERROR. If *STATUS* is TDS_DONE_CONTINUE, *CONN_OPTIONS* must be TDS_FLUSH.
TDS_DONE_ERROR (0x0002)	The last request received from the client resulted in an error.
TDS_DONE_COUNT (0x0010)	The *ROW_COUNT* argument contains a valid count value.

ROW_COUNT

(I) Number of rows selected or modified by the request. If this argument contains a valid number (a positive integer or zero), the *STATUS* argument should indicate TDS_DONE_COUNT. If the client request did not affect any rows (for example, it created or dropped a table), this argument does not contain a valid number, and TDS_DONE_COUNT should not be returned in the *STATUS* argument.

RETURN_STATUS_NUMBER

(I) Completion code used only with RPCs. An integer that is passed back to the client return status field to indicate normal completion, an error, or other condition. Sybase Adaptive Servers have predefined return status values for the numbers 0 and -1 to -14, listed in Table 3-38 on page 213. Values -15 to -99 are reserved for future use. To avoid conflict with Adaptive Server codes, use positive numbers for user-defined return status values.

The predefined Sybase return status values are listed in Table 3-38 on page 213. These values cannot be null.

Table 3-38: Predefined Sybase return values

Value	Meaning
0	Procedure executed without error.
-1	Missing object.
-2	Datatype error.

Value	Meaning
-3	Process was chosen as deadlock victim.
-4	Permission error.
-5	Syntax error.
-6	Miscellaneous user error.
-7	Resource error, such as out of space.
-8	Non-fatal internal problem.
-9	System limit was reached.
-10	Fatal internal inconsistency.
-11	Fatal internal inconsistency.
-12	Table or index is corrupt.
-13	Database is corrupt.
-14	Hardware error.

CONN_OPTIONS

(I) Connection open or closed indicator. Specifies whether the connection between the client and server should remain open or be closed.

Assign *CONN_OPTIONS* one of the following values:

TDS_ENDREPLY (1)	Indicates that the reply data stream ended. The communication state is changed from SEND to RECEIVE, and the transaction awaits the next request. When you use this value, *STATUS* must be TDS_DONE_FINAL. For IMS TM transactions, the TDSETPT *PROG_TYPE* argument must be EXPL. **Note** Select this option when using long-running transactions (CICS or explicit IMS TM only.) The IMS TM implicit API does not support long-running transactions.
TDS_ENDRPC (3)	Indicates that the data stream ended. This option ends the current conversation with the client and nullifies the handle specified in *TDPROC*. If a subsequent Gateway-Library function attempts to use that connection or handle, it results in an error or abend. When you use this value, *STATUS* must be TDS_DONE_FINAL.
TDS_FLUSH (7)	Indicates the end of a result set, but that another may follow. This option does not end the conversation, but it leaves the connection open. If *CONN_OPTIONS* is TDS_FLUSH, *STATUS* must be TDS_DONE_CONTINUE.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-39.

Table 3-39: TDSNDDON return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_STATUS (-174)	Invalid status value. The value entered in the *STATUS* field is invalid.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

Example 1

The following code fragment illustrates the use of TDINIT, TDACCEPT, TDSNDDON, and TDFREE at the beginning and end of a Gateway-Library program. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*-----------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-----------------------------------------------------------------*/

/*      ---------------------------------------------------------*/
/*      reset db2 error handlers                                 */
/*      ---------------------------------------------------------*/
       EXEC SQL WHENEVER SQLWARNING CONTINUE;
       EXEC SQL WHENEVER SQLERROR   CONTINUE;
       EXEC SQL WHENEVER NOT FOUND  CONTINUE;

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

```
  /*      establish gateway environment                                */
  /*      -------------------------------------------------------------*/
         CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);

  /*      -------------------------------------------------------------*/
  /*      accept client request                                        */
  /*      -------------------------------------------------------------*/
         CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                        SNA_CONNECTION_NAME,
                        SNA_SUBC);
  /*-------------------------------------------------------------------*/
  /*  (BODY OF PROGRAM)                                                */
  /*-------------------------------------------------------------------*/
         GO TO END_PROGRAM;

  /*-------------------------------------------------------------------*/
     END_PROGRAM:
  /*-------------------------------------------------------------------*/
         IF SEND_DONE = SEND_DONE_OK THEN
             WRK_DONE_STATUS = TDS_DONE_COUNT;

         ELSE
         DO;
             WRK_DONE_STATUS  = TDS_DONE_ERROR;
             PARM_RETURN_ROWS = 0;
         END;

         CALL TDSNDDON (GWL_PROC, GWL_RC,
                        WRK_DONE_STATUS,
                        PARM_RETURN_ROWS,
                        TDS_ZERO,
                        TDS_ENDRPC);

         CALL TDFREE (GWL_PROC, GWL_RC);
         EXEC CICS RETURN;

END SYSAMP1;
```

Example 2

The following code fragment illustrates the use of TDGETREQ in a program that uses the IMS TM implicit API. This example is taken from the sample program SYIPSAM1, listed in Appendix E, “Sample RPC Application for IMS TM (Implicit).”

```
  /*-------------------------------------------------------------------*/
  /*        WORK AREAS                                                 */
  /*-------------------------------------------------------------------*/
```

```
        DCL
            01  GW_LIB_MISC_FIELDS,
                05  GWL_SPA_PTR           PTR,
                05  GWL_PROC              PTR,
                05  GWL_INIT_HANDLE       PTR,
                05  GWL_RC                FIXED BIN(31),
                05  GWL_WAIT_OPTION       FIXED BIN(31),
                05  GWL_REQ_TYPE          FIXED BIN(31),
                05  GWL_PROG_TYPE         CHAR(04) INIT('MPP '),
                05  GWL_RPC_NAME          CHAR(30);
                 [rest of work areas]

/*      ------------------------------------------------------------*/
/*      set program type to MPP                                     */
/*      ------------------------------------------------------------*/
        CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                      GWL_SPA_PTR, TDS_NULL, TDS_NULL);
            .
            .   [check return code]
            .
/*      ------------------------------------------------------------*/
/*      accept first client request                                 */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME, SNA_SUBC);
            .
            .   [check return code; process request]
            .
DO WHILE(MORE_MSGS);
            .
            .   [prepare to send results; send reply rows]
            .
/*------------------------------------------------------------------*/
SEND_DONE:
/*------------------------------------------------------------------*/

        CALL TDSNDDON (GWL_PROC, GWL_RC, TDS_DONE_COUNT,
                       CTR_ROWS, TDS_ZERO, TDS_ENDRPC);
            .
            .   [check return code]
            .
        IF PARM_NR_ROWS = 0 THEN
            MORE_MSGS = FALSE;
        ELSE
           DO;
              GWL_WAIT_OPTION = TDS_TRUE;
```

```
                GWL_REQ_TYPE = 0;
                GWL_RPC_NAME = ' ';

                CALL TDGETREQ (GWL_PROC, GWL_RC,
                               GWL_WAIT_OPTION,
                               GWL_REQ_TYPE,
                               GWL_TRAN_NAME);

                SELECT (GWL_RC);
                  WHEN(TDS_OK);
                  WHEN(TDS_CONNECTION_TERMINATED)
                         MORE_MSGS = FALSE;
                  WHEN(TDS_RESULTS_COMPLETE)
                         MORE_MSGS = FALSE;
                  OTHERWISE
                      DO;
                         MORE_MSGS = FALSE;
                         CALL_NAME = 'TDGETREQ';
                         CALL DISP_ERROR;
                      END;
                 END;  /* SELECT */
              END;    /* END ELSE */

END;  /* DO WHILE MORE_MSGS = TRUE */
```

Example 3

The following code fragment illustrates the use of TDSNDDON and TDGETREQ in a program that uses the IMS TM explicit API. This example is taken from the sample program SYIXSAM1, listed in Appendix D, "Sample RPC Application for IMS TM (Explicit)."

```
/*-------------------------------------------------------------------*/
/*        WORK AREAS                                                 */
/*-------------------------------------------------------------------*/
      DCL
          01  GW_LIB_MISC_FIELDS,
              05  GWL_SPA_PTR           PTR,
              05  GWL_PROC              PTR,
              05  GWL_INIT_HANDLE       PTR,
              05  GWL_RC                FIXED BIN(31),
              05  GWL_WAIT_OPTION       FIXED BIN(31),
              05  GWL_REQ_TYPE          FIXED BIN(31),
              05  GWL_PROG_TYPE         CHAR(04) INIT('MPP '),
              05  GWL_TRAN_NAME         CHAR(30);
          .
          .   [rest of work areas]
          .
```

```
/*      ------------------------------------------------------------*/
/*      set program type to EXPL                                    */
/*      ------------------------------------------------------------*/
        GWL_PROG_TYPE = 'EXPL';

        CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                      GWL_SPA_PTR, TDS_NULL, TDS_NULL);
            .
            .   [check return code]
            .
/*      ------------------------------------------------------------*/
/*      accept first client request                                 */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME, SNA_SUBC);
            .
            .   [check return code; process request]
            .
DO WHILE(MORE_MSGS);
            .
            .   [prepare to send results; send reply rows]
/*------------------------------------------------------------------*/
  SEND_DONE:
/*------------------------------------------------------------------*/
        IF PARM_NR_ROWS = 0 THEN
           DO;
             MORE_MSGS = FALSE;
             GWL_SEND_DONE = TDS_ENDRPC;
           END;
        ELSE
           GWL_SEND_DONE = TDS_ENDREPLY;

        CALL TDSNDDON (GWL_PROC, GWL_RC, TDS_DONE_COUNT,
                       CTR_ROWS, TDS_ZERO, GWL_SEND_DONE);
            .   [check return code]
        IF PARM_NR_ROWS = 0 THEN
           DO;
              GWL_WAIT_OPTION = TDS_TRUE;
              GWL_REQ_TYPE = 0;
              GWL_RPC_NAME = ' ';

              CALL TDGETREQ (GWL_PROC, GWL_RC,
                             GWL_WAIT_OPTION,
                             GWL_REQ_TYPE,
                             GWL_TRAN_NAME);
```

```
                    SELECT (GWL_RC);
                      WHEN(TDS_OK);
                      WHEN(TDS_CONNECTION_TERMINATED)
                        DO;
                           MORE_MSGS = FALSE;
                           CALL FREE_STORAGE;
                        END;
                      WHEN(TDS_RESULTS_COMPLETE)
                        DO;
                           MORE_MSGS = FALSE;
                           CALL FREE_STORAGE;
                        END;
                      OTHERWISE
                        DO;
                           MORE_MSGS = FALSE;
                           CALL_NAME = 'TDGETREQ';
                           CALL DISP_ERROR;
                        END;
                    END;    /* SELECT */

            END;  /* IF */

        END;    /* DO WHILE MORE_MSGS = TRUE */
```

Usage

TDSNDDON

A server application uses this function to tell a client that it finished sending results and there is no additional data to be returned, or that an error or abnormal situation was detected by the server application. TDSNDDON also indicates whether the client/server connection should remain open or be closed.

- When *STATUS* is TDS_DONE_FINAL, TDSNDDON sends return parameter information back to the client. The return parameter value *must* be previously set by TDSETPRM.
- When the connection remains open, this function puts the server application into RECEIVE state to await another request. In this case, that application should call TDRESULT next, to determine the client response.
- The application must be in SEND state for this function to execute successfully. If it is not in SEND state, TDSNDDON returns TDS_WRONG_STATE. Call TDRESULT to put your application in SEND state.
- See the discussion of RETURN in the Adaptive Server Enterprise *Reference Manual* for more information about return status values.

- This call controls whether the connection between a client and a server should remain open or whether it should be closed.
- To prepare to accept additional client requests after all results are returned, set *STATUS* to TDS_DONE_FINAL and *CONN_OPTIONS* to TDS_ENDREPLY. Then, call TDGETREQ to accept the next client request.
- A return code of TDS_CANCEL_RECEIVED indicates that the client sent an ATTENTION. Once it receives an ATTENTION, Open ServerConnect does not forward any results to the client.

  All Open ServerConnect application programs should check for TDS_CANCEL_RECEIVED frequently, and send a TDSNDDON as soon as possible after one is received.

Note If a client ATTENTION is received *after* all results are sent by the Gateway-Library transaction, Open ServerConnect may forward results to the client before it is aware that the client canceled the request.

For long-running transactions

The transaction stays active and processes new requests as they are sent.

Note IMS TM users: Long-running transactions are only supported for the explicit API (the TDSETPT *PROG_TYPE* parameter is set to EXPL).

For short-running transactions

A transaction ends after it sends results to the client.

For Japanese users

The JCM converts the data in the return parameter from mainframe to workstation before sending it back to the client.

See also

Related functions

- TDACCEPT on page 68
- TDRESULT on page 174
- TDSETPRM on page 194

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*
- Adaptive Server Enterprise *Reference Manual* (for a discussion of return status values)

TDSNDMSG

Description

Sends an error or informational message to the client.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC              PTR,
01 RETCODE             FIXED BIN(31),
01 MESSAGE_TYPE        FIXED BIN(31),
01 MESSAGE_NUMBER      FIXED BIN(31),
01 SEVERITY            FIXED BIN(31),
01 ERROR_STATE         FIXED BIN(31),
01 LINE_ID             FIXED BIN(31),
01 TRANSACTION_ID      CHAR(n),
01 TRANSACTION_ID_LENGTH FIXED BIN(31),
01 MESSAGE_TEXT        CHAR(n),
01 MESSAGE_LENGTH      FIXED BIN(31);

CALL TDSNDMSG(TDPROC,RETCODE, MESSAGE_TYPE,
          MESSAGE_NUMBER, SEVERITY, ERROR_STATE,
          LINE_ID, TRANSACTION_ID,
          TRANSACTION_ID_LENGTH, MESSAGE_TEXT,
          MESSAGE_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-40 on page 224.

MESSAGE_TYPE

(I) Category of message being sent. Indicates whether it is an informational message or an error message. Assign this argument one of the following values:

TDS_INFO_MSG (1)	Message is an informational message.
TDS_ERROR_MSG (2)	Message is an error message.

MESSAGE_NUMBER

(I) Message number. This value is always four bytes in length. Where possible, use Sybase-compatible error numbers.

For messages sent to Open Client programs, this value is stored in the SMSG_NO field of the Open Client *CS_SERVERMSG* structure.

SEVERITY

(I) Severity level of the error. A value of 10 or less represents an informational message.

For messages sent to Open Client clients, this value is stored in the SMSG_SEV field of the Open Client *CS_SERVERMSG* structure.

Specify one of the following severity values:

TDS_INFO_SEV (0)	Informational message
TDS_ERROR_SEV (10)	Error message

ERROR_STATE

(I) Error state number. This number provides additional information about the context of the error.

For messages sent to Open Client clients, this value is stored in the SMSG_STATE field of the Open Client *CS_SERVERMSG* structure.

LINE_ID

(I) An additional identifier assigned by the program. You determine how to use this argument at your site.

For messages sent to Open Client clients, this value is stored in the SMSG_LINE field of the Open Client *CS_SERVERMSG* structure.

TRANSACTION_ID

(I) Identifier of the transaction that is currently executing. The transaction ID identifies the transaction that is issuing the error message.

Under CICS: This is the *TRANSID* from the CICS Program Control Table (PCT).

Under IMS TM: This is the transaction name defined when the system is generated.

Under MVS: This is the APPC transaction name defined in the transaction profile.

TRANSACTION_ID_LENGTH

(I) Length of the *TRANSACTION_ID*. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

Under CICS: For CICS Version 1.7, this value is always 4 or less. For later versions, it is the actual length of the transaction ID, which can be greater than 4.

Under IMS TM: This value is always 8 or less.

Under MVS: This is the APPC transaction name defined in the transaction profile. This value is normally 8 or less.

MESSAGE_TEXT

(I) The text of the message.

For messages sent to Open Client clients, this value is stored in the SMSG_TEXT field of the Open Client *CS_SERVERMSG* structure.

MESSAGE_LENGTH

(I) Length of the message text. The maximum permitted length for a message is 512 bytes.

If you are using the Japanese Conversion Module (JCM), it adjusts this length to the length used by the client character set.

For messages sent to Open Client clients, this value is stored in the *SMSG_TEXT_LEN* field of the Open Client *CS_SERVERMSG* structure.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-40.

Table 3-40: TDSNDMSG return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).

Return value	Meaning
TDS_INVALID_DATA_TYPE (-171)	Illegal datatype. A Sybase datatype supplied in the call is not supported and the conversion cannot be done. The operation failed.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *MESSAGE_LENGTH* argument is too long.
TDS_INVALID_NAMELENGTH (-179)	Invalid name length. The length specified for the column, parameter, message, or server name is invalid.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_STATUS (-174)	Invalid status value. The value entered in the *STATUS* field is invalid.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates a typical use of TDSNDMSG. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*-------------------------------------------------------------------*/
SEND_ERROR_MESSAGE: PROC;
/*-------------------------------------------------------------------*/
       SEND_DONE    = SEND_DONE_ERROR;
       MSG_SEVERITY = MSG_SEVERITY_ERROR;
       MSG_NR       = MSG_NR_ERROR;
       MSG_TYPE     = TDS_ERROR_MSG;
       CALL SEND_MESSAGE;

       RETURN;

END SEND_ERROR_MESSAGE;
/*-------------------------------------------------------------------*/
   SEND_MESSAGE: PROC;
/*-------------------------------------------------------------------*/
/*      -------------------------------------------------------------*/
```

```
      /*      ensure we're in right state to send a message                */
      /*      ------------------------------------------------------------*/
              CALL TDSTATUS (GWL_PROC, GWL_RC,
                             GWL_STATUS_NR,
                             GWL_STATUS_DONE,
                             GWL_STATUS_COUNT,
                             GWL_STATUS_COMM,
                             GWL_STATUS_RETURN_CODE,
                             GWL_STATUS_SUBCODE);

              IF GWL_RC = TDS_OK THEN
              DO;
                  IF GWL_STATUS_COMM = TDS_RECEIVE THEN

                  CALL TDSNDMSG (GWL_PROC, GWL_RC,
                                 MSG_TYPE, MSG_NR,
                                 MSG_SEVERITY,
                                 TDS_ZERO,
                                 TDS_ZERO,
                                 MSG_RPC, MSG_RPC_L,
                                 MSG_TEXT, MSG_TEXT_L);
              END;
              RETURN;
      END SEND_MESSAGE;
```

Usage

TDSNDMSG

A server application uses this function to send an error or informational message to a remote client.

Note IMS TM Users: The term “message” is used here in the narrow sense of error or informational messages sent to the client; it is not used in the IMS TM sense of message processing.

- Errors related to the operation of the TRS are recorded in its error log, available to the TRS administrator. Errors related to the client program are passed on to the requesting client. A client handles an Open ServerConnect error message like any error returned by Adaptive Server.
- Messages can be sent before a row is described or after all rows are sent. An application can call TDSNDMSG either before a TDESCRIB or after the last TDSNDROW call for the described row. No messages can be sent between a TDESCRIB and a TDSNDROW or between two TDSNDROW calls.

- Your application must be in SEND state for this function to execute successfully. If it is not in SEND state, TDSNDMSG returns TDS_WRONG_STATE. Call TDRESULT to put your application in SEND state.
- A transaction can send a message to a client after TDSNDDON only if the value of the TDSNDDON argument *STATUS* is TDS_DONE_CONTINUE, and the value of *CONN_OPTIONS* is TDS_FLUSH. If the value of *CONN_OPTIONS* is TDS_ENDRPC or TDS_ENDREPLY, no messages can be sent after the TDSNDDON call is issued.
- For more information about errors and error severity levels, see Mainframe Connect Client Option and Server Option *Messages and Codes*.

For Japanese users

If the Japanese Conversion Module (JCM) is used, TDSNDMSG converts the message data from the mainframe character set to the workstation character set and adjusts the message length before sending, if necessary.

See also

Related documents

- Open Client DB-Library *Reference Manual* (dbmsghandle)
- Mainframe Connect Client Option and Server Option *Messages and Codes*
- Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*

TDSNDROW

Description

Sends a row of data back to the requesting client, over the specified connection.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC  PTR,
01 RETCODE FIXED BIN(31);

CALL TDSNDROW  (TDPROC, RETCODE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-41.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-41.

Table 3-41: TDSNDROW return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_CONNECTION_FAILED (-4998)	Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).
TDS_CONNECTION_TERMINATED (-4997)	Connection closed. The remote partner closed (deallocated) the client/server connection.
TDS_DATE_CONVERSION_ERROR (-23)	Error in conversion of datetime data. This can be a result of trying to convert short datetime (TDSDATETIME4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short datetime datatype.
TDS_DECIMAL_CONVERSION_ERROR (-24)	Error in conversion of packed decimal data.
TDS_FLOAT_CONVERSION_ERROR (-21)	Error in conversion of float values.
TDS_ILLEGAL_REQUEST (-5)	Illegal function. The operation failed. This code can indicate that a client application is trying to use a Gateway-Library function that is not supported for clients (for example, TDSNDROW).
TDS_INVALID_LENGTH (-173)	The length specified in the preceding TDESCRIBE is wrong.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_MONEY_CONVERSION_ERROR (-22)	Error in conversion of TDSMONEY-type data. This can be a result of trying to convert to short money (TDSMONEY4) for a client using an early TDS version. TDS versions earlier than 4.2 do not support the short money datatype.
TDS_TRUNCATION_ERROR (-20)	Error occurred in truncation of data value.

Return value	Meaning
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following code fragment illustrates a typical use of TDSNDROW. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
/*-------------------------------------------------------------------*/
SEND_ROWS:
/*-------------------------------------------------------------------*/
       DO WHILE(^ ALL_DONE);
          CALL FETCH_AND_SEND_ROWS;
       END;
/*-------------------------------------------------------------------*/
FETCH_AND_SEND_ROWS: PROC;
/*-------------------------------------------------------------------*/
       EXEC SQL FETCH ECURSOR INTO :EMPLOYEE_FIELDS;

       IF SQLCODE = 0 THEN
       DO;
/*           ---------------------------------------------------------*/
/*           Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.    */
/*           ---------------------------------------------------------*/
            WRKLEN1 = STG(EMPLOYEE_SAL);
            WRKLEN2 = STG(WRK_EMPLOYEE_SAL);
CALL TDCONVRT (GWL_PROC, GWL_RC,
                           GWL_CONVRT_SCALE,
                           TDSDECIMAL,
                           WRKLEN1,
                           EMPLOYEE_SAL,
                           TDSMONEY,
                           WRKLEN2,
                           WRK_EMPLOYEE_SAL);
/*           ---------------------------------------------------------*/
/*           Do not send trailing blanks of EMPLOYEE_LNM              */
/*           ---------------------------------------------------------*/
            WRKLEN1 = LENGTH(EMPLOYEE_LNM);
            CTR_COLUMN = 2;
  WRK_BLANKS_SS = 1;
LOOP:       DO WHILE(WRK_BLANKS_SS <= WRKLEN1);
              IF SUBSTR(EMPLOYEE_LNM, WRK_BLANKS_SS, 1) = ' ' THEN DO;
                LEAVE LOOP;
              END;
```

```
                WRK_BLANKS_SS = WRK_BLANKS_SS + 1;
              END LOOP;

              IF (WRK_BLANKS_SS <= WRKLEN1) THEN DO;
                CALL TDSETLEN (GWL_PROC, GWL_RC,
                               CTR_COLUMN,
                               WRK_BLANKS_SS - 1);
              END;
/*            ------------------------------------------------------*/
/*            send a row to the client                              */
/*            ------------------------------------------------------*/
              CALL TDSNDROW (GWL_PROC, GWL_RC);
              PARM_RETURN_ROWS = PARM_RETURN_ROWS + 1;

              IF GWL_RC = TDS_CANCEL_RECEIVED THEN
              DO;
                 ALL_DONE = ALL_DONE_YES;
              END;
          END;

          ELSE IF SQLCODE = +100 THEN
              DO;
                 ALL_DONE = ALL_DONE_YES;
              END;

          ELSE IF SQLCODE < 0 THEN
          DO;
              ALL_DONE = ALL_DONE_YES;
              CALL FETCH_ERROR;
          END;
      RETURN;
      END FETCH_AND_SEND_ROWS;
```

Usage

TDSNDROW

A server application uses this function to send a row of data to the requesting client over the connection specified in *TDPROC*. Each TDSNDROW sends a single row, so the application must issue a TDSNDROW call for each row to be sent.

- TDSNDROW sends the column name and format before it sends the column data.

Note If your IMS TM transaction is conversational (CONV), you must insert the scratch pad area at the beginning of the IO/PCB before sending the results with TDSNDROW.

- A server application cannot send any data rows to the client after it issues TDSNDMSG or TDSNDDON, unless the TDSNDDON status is TDS_DONE_CONTINUE.
- Before a row of data can be sent to a client, each column of the row must be defined in a TDESCRIB call. If your application calls TDSNDROW before all the columns in the row are described with TDESCRIB, this function returns TDS_WRONG_STATE, and the row is not sent.
- If the column datatype is TDSVARYCHAR, TDSVARYBIN, or TDSVARYGRAPHIC, the column length is determined each time a row is sent by the value of the "LL" specification at the beginning of the column structure.

Datatype conversions

Table 3-42 shows the conversions that TDSNDROW performs.

Table 3-42: Datatype conversions performed by TDSNDROW

Source datatype: Gateway-Library	Result datatype: Open Client	Notes
TDSCHAR	TDSVARYCHAR	Performs EBCDIC and ASCII conversion. For Japanese character sets, does mainframe to workstation conversion. Pads TDSCHAR fields with blanks.
TDSCHAR	TDSMONEY	
TDSVARYCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSMONEY	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	
TDSFLT4	TDSFLT8	
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSCHAR	TDSMONEY	
TDSVARYCHAR	TDSMONEY	
TDS_PACKED_DECIMAL	TDSCHAR	When converting packed decimal to character values, change the length to allow for unpacking, leading or trailing zeros, the sign, and the decimal point.
TDS_PACKED_DECIMAL	TDSVARYCHAR	
TDS_PACKED_DECIMAL	TDSMONEY	
TDS_PACKED_DECIMAL	TDSFLT4	
TDS_PACKED_DECIMAL	TDSFLT8	

Source datatype: Gateway-Library	Result datatype: Open Client	Notes
TDS_PACKED_DECIMAL TDS CHAR	TDSNUMERIC TDSNUMERIC	Use TDSETBCD after TDESCRIB to set precision and scale for numeric and Sybase decimal columns.
TDS_PACKED_DECIMAL TDSCHAR	TDS_SYBASE_DECIMAL TDS_SYBASE_DECIMAL	Use TDSETBCD after TDESCRIB to set precision and scale for numeric and Sybase decimal columns.
TDSGRAPHIC TDSGRAPHIC TDSVARYGRAPHIC TDSVARYGRAPHIC	TDSCHAR TDSVARYCHAR TDSCHAR TDSVARYCHAR	Performed by Japanese Conversion Module (JCM). Pads TDSCHAR fields with blanks.
TDSDATETIME TDSDATETIME4	TDSCHAR TDSCHAR	

- Your application must be in SEND state for this function to execute successfully. If it is not in SEND state, TDSNDROW returns TDS_WRONG_STATE. Call TDRESULT to put your application in SEND state.
- If the *RETCODE* argument contains the value, TDS_CANCEL_RECEIVED, your application should immediately stop sending rows and issue TDSNDDON and TDFREE. It is a good idea to check the return code after each row is sent.

For Japanese users

If the JCM is used, TDSNDROW converts the data in a column from the mainframe character set to the workstation character set before sending, if necessary.

See also

Related functions

- TDESCRIB on page 89
- TDSNDDON on page 212
- TDSNDMSG on page 222

TDSQLLEN

Description

Determines the length of a language string received from a client.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC     PTR,
01 SQL_LENGTH FIXED BIN(31);

CALL TDSQLLEN  (TDPROC, SQL_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

SQL_LENGTH

(O) The length of the incoming language string. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

Return value

This function has no *RETCODE* argument. It returns the length of the SQL string in the *SQL_LENGTH* argument. If the value in *SQL_LENGTH* is -1, call TDRESULT, and examine its return code to determine what the problem is.

Examples

The following code fragment illustrates the use of TDSQLLEN to determine the length of the incoming SQL string. This example is taken from the sample program in Appendix C, “Sample Language Application for CICS.”

```
/*-------------------------------------------------------------------*/
READ_IN_SQL_TEXT:
/*-------------------------------------------------------------------*/

/*      -------------------------------------------------------------*/
/*      prepare for receive                                          */
/*      -------------------------------------------------------------*/
       CALL TDRESULT (GWL_PROC, GWL_RC);

/*      -------------------------------------------------------------*/
/*      load ptr to redefined sql text                               */
/*      -------------------------------------------------------------*/
       LANG_BUFFER_PTR = ADDR(LANG_BUFFER_DB2);

/*      -------------------------------------------------------------*/
/*      get len of language text, ensure not too big for us          */
/*      (this could be handled without TDSQLLEN by checking          */
/*      LANG_ACTUAL_LEN doesn't exceed LANG_MAX_L in TDRCVSQL call)  */
/*      -------------------------------------------------------------*/
       CALL TDSQLLEN (GWL_PROC, GWL_SQLLEN);

       LANG_MAX_L = STG(LANG_BUFFER_TEXT);
```

```
            IF GWL_SQLLEN > LANG_MAX_L THEN
            DO;
                MSG_TEXT   = MSG_BAD_LEN;
                MSG_TEXT_L = STG(MSG_BAD_LEN);
                CALL SEND_ERROR_MESSAGE;
                GO TO END_PROGRAM;
            END;

    /*      ------------------------------------------------------------*/
    /*      get language text                                           */
    /*      ------------------------------------------------------------*/
            CALL TDRCVSQL (GWL_PROC, GWL_RC,
                           LANG_BUFFER_TEXT,
                           LANG_MAX_L,
                           LANG_ACTUAL_L);

            LANG_BUFFER_LL = LANG_ACTUAL_L;
```

Usage

TDSTATUS

A server application uses this function to determine the actual length of an incoming string.

- Typically, an application calls TDSQLLEN after TDACCEPT and before TDRCVSQL to determine how large a storage area to allocate for the incoming string.
- You can use TDSQLLEN to store incoming text in more than one variable. Read part of the text into one variable, using TDRCVSQL, then call TDSQLLEN to determine the length of the text that remains. Call TDRCVSQL again to move the remaining text into a second variable. Repeat as often as necessary.
- Although this function is called TDSQLLEN, it does not differentiate between SQL strings and other language strings, such as math functions or single-byte katakana. It is up to the application to determine what kind of text is in the buffer and what to do with it.

Related functions

- TDRCVSQL on page 169

TDSTATUS

Description

Retrieves the last status information received from a remote procedure call (RPC) or SQL command string.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC              PTR,
01 RETCODE             FIXED BIN(31),
01 RETURN_STATUS_NUMBER FIXED BIN(31),
01 DONE_STATUS         FIXED BIN(31),
01 DONE_COUNT          FIXED BIN(31),
01 COMM_STATE          FIXED BIN(31),
01 COMM_RETCODE        FIXED BIN(31),
01 COMM_ERROR_SUBCODE  FIXED BIN(31);

CALL TDSTATUS(TDPROC, RETCODE, RETURN_STATUS_NUMBER,
              DONE_STATUS, DONE_COUNT, COMM_STATE,
              COMM_RETCODE, COMM_ERROR_SUBCODE);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-43 on page 236.

RETURN_STATUS_NUMBER

(O) Variable where the completion code for this request is stored. This code is an integer that indicates normal completion, an error, or other condition. Negative numbers (-1 to -99) and zero are Sybase-defined return status numbers. Positive numbers are user-defined values. For a list of Sybase-defined return status numbers, see the discussion of TDSNDDON on page 212.

DONE_STATUS

(O) Variable where the result of the operation is stored. This value indicates whether the operation completed normally or returned an error, and whether any rows were affected. *DONE_STATUS* returns one of the following values:

TDS_DONE_FINAL (0x0000)	The set of results currently being sent is the final set of results.
TDS_DONE_CONTINUE (0x0001)	More results follow. This option tells the receiving program to continue retrieving results until this argument specifies TDS_DONE_FINAL or TDS_DONE_ERROR.

TDS_DONE_ERROR (0x0002)	The last request received from the client resulted in an error.
TDS_DONE_COUNT (0x0010)	The *ROW_COUNT* argument contains a valid count value.

DONE_COUNT

(O) Variable where the row count for the operation is stored. If the *DONE_STATUS* indicates that a valid number of rows was affected by the operation, this value indicates how many rows were affected.

COMM_STATE

(O) Variable where the current communication state of the mainframe server is stored. *COMM_STATE* returns one of the following values:

TDS_RESET (0)	Client/server conversation for this transaction ended. If the current transaction is running under CICS or uses the IMS TM explicit API, the transaction should exit as soon as possible. If the current transaction is a WFI transaction using the IMS TM implicit API, the transaction can accept another client request by calling TDGETREQ.
TDS_SEND (1)	Transaction is in SEND state.
TDS_RECEIVE (2)	Transaction is in RECEIVE state.

TDINFRPC also returns this information.

See "Communication states" on page 19 for an explanation of communication states.

COMM_RETCODE

(O) Variable where the TDPROC current communication I/O return code is stored. This value is in SAA format.

COMM_ERROR_SUBCODE

(O) Detailed error information. Provides additional information about the cause of failure when TDSTATUS returns a return code other than TDS_OK.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-43.

Table 3-43: TDSTATUS return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_CANCEL_RECEIVED (-12)	Operation canceled. The remote partner issued a cancel. The current operation failed.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Return value	Meaning
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_STATUS_NOT_RECEIVED (-11)	No status returned from client. No *RETURN_STATUS_NUMBER* is available because the server did not yet send the status back to the client.

Examples

The following code fragment illustrates the use of TDSTATUS to check the communication state before sending a message. This example is taken from the sample program in Appendix B, "Sample RPC Application for CICS."

```
  /*-------------------------------------------------------------------*/
  SEND_ERROR_MESSAGE: PROC;
  /*-------------------------------------------------------------------*/
        SEND_DONE    = SEND_DONE_ERROR;
        MSG_SEVERITY = MSG_SEVERITY_ERROR;
        MSG_NR       = MSG_NR_ERROR;
        MSG_TYPE     = TDS_ERROR_MSG;
        CALL SEND_MESSAGE;

        RETURN;

END SEND_ERROR_MESSAGE;
  /*-------------------------------------------------------------------*/
  SEND_MESSAGE: PROC;
  /*-------------------------------------------------------------------*/

  /*      ---------------------------------------------------------*/
  /*      ensure we're in right state to send a message             */
  /*      ---------------------------------------------------------*/
        CALL TDSTATUS (GWL_PROC, GWL_RC,
                       GWL_STATUS_NR,
                       GWL_STATUS_DONE,
                       GWL_STATUS_COUNT,
                       GWL_STATUS_COMM,
                       GWL_STATUS_RETURN_CODE,
                       GWL_STATUS_SUBCODE);

        IF GWL_RC = TDS_OK THEN
        DO;
            IF GWL_STATUS_COMM = TDS_RECEIVE THEN

            CALL TDSNDMSG (GWL_PROC, GWL_RC,
                           MSG_TYPE, MSG_NR,
                           MSG_SEVERITY,
                           TDS_ZERO,
```

```
                                TDS_ZERO,
                                MSG_RPC, MSG_RPC_L,
                                MSG_TEXT, MSG_TEXT_L);
          END;
          RETURN;
     END SEND_MESSAGE;
```

Usage

TDSTATUS

This function returns the TDS status number, status flags, count, and communication state associated with the current RPC or SQL command batch execution.

- TDSTATUS returns standard communication subcodes. These codes are listed in Mainframe Connect Client Option and Server Option *Messages and Codes*.

See also

Related functions

- TDRESULT on page 174
- TDSNDDON on page 212

Related topics

- "Long-running transactions" on page 52

Related documents

- Mainframe Connect Client Option and Server Option *Messages and Codes*

TDTERM

Description

Frees up all MVS storage.

Syntax

```
% INCLUDE SYGWPLI;

01 IHANDLE PTR,
01 RETCODE FIXED BIN(31);

CALL TDTERM (IHANDLE, RETCODE);
```

Parameters

IHANDLE

(I) A transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same *IHANDLE* specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-44.

Return value

The *RETCODE* argument can contain any of the return values in the following table.

Table 3-44: TDTERM return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_INVALID_IHANDLE (-19)	Invalid *IHANDLE* specification. Error in specifying a value for the *IHANDLE* argument.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.

Examples

The following code fragment illustrates the use of TDFREE and TDTERM at the end of a Gateway-Library program. This example is taken from the sample program SYIXSAM1, listed in Appendix E, "Sample RPC Application for IMS TM (Implicit)."

```
FREE_STORAGE: PROC;

          CALL TDFREE (GWL_PROC, GWL_RC);

          IF GWL_RC ^= 0 THEN
             DO;
                CALL_NAME = 'TDFREE  ';
                CALL_MSG = 'ERROR IN CALL RC=';
                CALL_RC  = GWL_RC;
                PUT FILE(SYSPRINT) DATA(ERROR_MSG);
             END;

          CALL TDTERM (GWL_INIT_HANDLE, GWL_RC);

          IF GWL_RC ^= 0 THEN
             DO;
                CALL_NAME = 'TDTERM  ';
                CALL_MSG = 'ERROR IN CALL RC=';
                CALL_RC  = GWL_RC;
                PUT FILE(SYSPRINT) DATA(ERROR_MSG);
```

```
            END;

    END FREE_STORAGE;
```

Usage

TDTERM

This function frees all TDPROCs and underlying free control blocks, including the IHANDLE, in preparation for program termination. It also deallocates the connection, if it is still active, and frees its queues.

- This function is required in MVS and IMS TM programs to free up MVS storage when the MVS or IMS TM application exits. In CICS programs, it is optional.

See also

Related functions

- TDACCEPT on page 68
- TDGETREQ on page 103
- TDINIT on page 148
- TDSETPT on page 200

Related topics

- “Long-running transactions” on page 52

Related documentation

- Mainframe Connect Server Option for IMS and MVS *Installation and Administration Guide*

TDYNAMIC

Description

Reads or responds to a client dynamic SQL command.

Syntax

```
%INCLUDE SYGWPLI;

01 TDPROC           POINTER,
01 RETCODE          FIXED BIN (31),
01 CMD              FIXED BIN (31),
01 ITEM             FIXED BIN (31),
01 HOST-VARIABLE    CHAR(n),
01 HOST-VAR-LENGTH  FIXED BIN (31),
01 ACTUAL-DATA-LENGTH FIXED BIN (31);
```

```
CALL TDYNAMIC (TDPROC, RETCODE, CMD, ITEM,
        HOST_VARIABLE, HOST_VAR_LENGTH,
         ACTUAL_DATA_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-46 on page 242.

CMD

(I) Gets (TDS_GET) or sets (TDS_SET) the value of a particular item.

ITEM

(I/O) Indicates what kind of information is being sent or retrieved.

The valid values are listed in Table 3-45.

Table 3-45: ITEM argument values

Value	Notes
TDS_DYN_TYPE	Legal values for TDS_DYN_TYPE when *CMD* is TDS_GET: TDS_PREPARE TDS_DESCRIBE_INPUT TDS_DESCRIBE_OUTPUT TDS_EXECUTE TDS_EXEC_IMMEDIATE TDS_DEALLOC When *CMD* is TDS_SET, TDS_DYN_ACK is the only valid value.
TDS_DYN_IDLEN	The length of the dynamic statement ID being sent or retrieved.
TDS_DYN_ID	The dynamic statement ID being sent or retrieved.
TDS_DYN_STMTLEN	The length of the dynamic statement being sent or retrieved.
TDS_DYN_STMT	The dynamic statement that is being prepared or executed.

HOST_VARIABLE

(I/O) Buffer in which *ITEM* value is returned (TDS_GET) or set (TDS_SET).

HOST_VAR_LEN

(I/O) The length in bytes of *HOST_VARIABLE*. This value varies depending on the value of *ITEM*. The possible values are:

Value	Notes
TDS_DLYN_TYPE (4)	
TDS_DYN_IDLEN (4)	
TDS_DYN_ID	Depends on the size of ID (maximum is 255)
TDS_DYN_STMTLEN (4)	
TDS_DYN_STMT	Depends on the size of dynamic statement

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-46.

Table 3-46: TDYNAMIC return values

Return value	Meaning
TDS_DYNSQL_ALREADY_DEALLOCATED (-84)	Dynamic SQL request already allocated. You cannot deallocate a dynamic SQL request that is already allocated.
TDS_DYNSQL_ALREADY_PREPARED (-81)	Dynamic SQL request already prepared. You cannot prepare a dynamic SQL request that is already deallocated.
TDS_DYNSQL_ID_NOT_FOUND (-85)	Dynamic SQL request not found.
TDS_DYNSQL_IDLEN_TOO_LONG (-87)	Dynamic SQL request ID length is greater than 255.
TDS_DYNSQL_NO_STMT_GIVEN (-86)	No SQL statement is associated with the dynamic SQL request.
TDS_DYNSQL_NOT_PREPARED (-80)	A dynamic SQL request is not prepared.
TDS_DYNSQL_OUTPUT_ALREADY_DEFINED (-83)	Dynamic SQL output already defined. You cannot define dynamic SQL output more than once.
TDS_DYNSQL_PARMS_ALREADY_DEFINED (-82)	Dynamic SQL parameters already defined. You cannot define dynamic SQL parameters more than once.
TDS_DYNSQL_STMT_NOT_FOUND (-89)	No SQL statement is associated with the dynamic SQL request.
TDS_INVALID_BOOLEAN (-180)	Invalid Boolean value. Boolean values must be set to either CS_TRUE or CS_FALSE.

Return value	Meaning
TDS_INVALID_CURCLOSOPTION (-182)	A “closed” cursor command specified an invalid option. The Gateway-Library transaction received a “closed” cursor command, but the value of the OPTION field of the *CURSOR_DESC* structure is invalid. Valid options are TDS_CUR_UNUSED and TDS_CUR_DEALLOC.
TDS_INVALID_CURDECLOPTION (-183)	A declare cursor command has an invalid option specified. The Gateway-Library transaction received a declare cursor command, but the value of the OPTION field of the *CURSOR_DESC* structure is invalid. Valid options are TDS_CUR_UNUSED and TDS_CUR_DEALLOC.
TDS_INVALID_CURDECLSTAT (-184)	Illegal cursor declare option.
TDS_INVALID_CURINFCMD (-195)	Illegal cursor information command.
TDS_INVALID_CUROPENSTAT (-187)	Illegal cursor open status.
TDS_INVALID_CURUPDSTAT (-186)	Illegal cursor update status.
TDS_INVALID_DATA_CONVERSION (-172)	Incompatible datatypes. The source datatype cannot be converted into the requested result datatype.
TDS_INVALID_DATA_TYPE (-171)	Illegal datatype. A Sybase datatype supplied in the call is not supported and the conversion cannot be done. The operation failed.
TDS_INVALID_DATAFMT_VALUE (-181)	One or more values specified for fields in the *DATAFMT* structure are illegal.
TDS_INVALID_DYNSQL_FSM (-79)	Dynamic SQL request in invalid state.
TDS_INVALID_DYNSTAT (-188)	Invalid status for dynamic SQL request.
TDS_INVALID_DYNTYPE (-189)	Invalid type for dynamic SQL request.
TDS_INVALID_FLAGS (-176)	Invalid padding option for a field.
TDS_INVALID_LENGTH (-173)	Wrong length. The length specified in the *HOST_VAR_LEN* argument is too long.
TDS_INVALID_NAMELENGTH (-179)	Invalid name length. The length specified for the column, parameter, message, or server name is invalid.
TDS_INVALID_PRECISION (-177)	Invalid precision value. The precision value specified during conversion of TDS_PACKED_DECIMAL data is invalid.
TDS_INVALID_SCALE (-178)	Invalid scale value. The scale value specified during conversion of TDS_PACKED_DECIMAL data is invalid.

Return value	Meaning
TDS_INVALID_STATUS (-174)	Invalid status value. The value entered in the *STATUS* field is invalid.
TDS_INVALID_VAR_ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A NULL value was specified. The operation failed.

TDWRTLOG

Description

Writes a user-created message or a system entry to the trace and error log.

Syntax

```
% INCLUDE SYGWPLI;

01 TDPROC        PTR,
01 RETCODE       FIXED BIN(31),
01 DATETIME_FLAG  FIXED BIN(31),
01 MESSAGE       CHAR(n),
01 MESSAGE_LENGTH FIXED BIN(31);

CALL TDWRTLOG(TDPROC, RETCODE, DATETIME_FLAG,
              MESSAGE, MESSAGE_LENGTH);
```

Parameters

TDPROC

(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The *TDPROC* handle corresponds to the connection and command handles in Open Client Client-Library.

RETCODE

(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-47 on page 245.

DATETIME_FLAG

(I) Timestamp indicator. This flag indicates whether or not the log message should begin with the current date and time. Assign this argument one of the following values:

TDS_TRUE (1)	Include date and time.
TDS_FALSE (0)	Do not include date and time.

MESSAGE

(I) User-written message. This is the text of the message to be written to the trace file.

MESSAGE_LENGTH

(I) Length of the user-written message. The message must be less than or equal to 80 bytes in length. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.

Return value

The *RETCODE* argument can contain any of the return values listed in Table 3-47.

Table 3-47: TDWRTLOG return values

Return value	Meaning
TDS_OK (0)	Function completed successfully.
TDS_LOG_ERROR(-258)	Attempt to write to the log file failed.
TDS_INVALID_PARAMETER (-4)	Invalid parameter value. The value assigned to one or more of the arguments supplied in the call is not valid. The operation failed.
TDS_INVALID_TDPROC (-18)	Error in specifying a value for the *TDPROC* argument.
TDS_WRONG_STATE (-6)	This function cannot be used in the current communication state. For example, your program tried to send a reply before it read in all of the client parameters. The application was still in RECEIVE state and could not send. The operation failed.

Examples

The following example writes an entry to the trace log after checking to see that tracing is enabled. It is taken from the sample program in Appendix G, "Sample Tracing and Accounting Program," which runs under CICS.

```
/*--------------------------------------------------------------------*/
    TDWRTLOG_PROC: PROC;
    /*--------------------------------------------------------------------*/
    /*      ------------------------------------------------------------*/
    /*      Write a log entry only if logging is enabled.                 */
    /*      ------------------------------------------------------------*/
          CALL GET_TRACE_STATUS;
          IF (TRACING) THEN
          DO;
              CALL TDWRTLOG (GWL_PROC, GWL_RC,
                             TDS_TRUE,
                             GWL_WRTLOG_MSG,
                             GWL_WRTLOG_MSG_L);
              IF GWL_RC NE TDS_OK THEN
              DO;
                  SEND_DONE       = SEND_DONE_ERROR;
                  MSG_SRVLIB_FUNC = 'TDWRTLOG';
                  GO TO TDWRTLOG_EXIT;
              END;
```

```
        END;
        ELSE
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'LOGNOTON';
            GO TO TDWRTLOG_EXIT;
        END;
/*-------------------------------------------------------------------*/
TDWRTLOG_EXIT:
/*-------------------------------------------------------------------*/
        RETURN;
END TDWRTLOG_PROC;
```

Usage

TDWRTLOG

You use this function to write a message to the trace and error log.

- Traces and error messages are written to the same log. The transaction processing system determines the log used for tracing and the type of tracing being done. Also, the log must be open for this function to execute successfully.
  - *Under CICS.* The trace and error log is a VSAM ESDS file.

    As installed, the CICS trace log is named *SYTDLOG1*. You can change the name of this file with TDSETLOG. To find out the name of the current trace log, use TDINFLOG.

    TDSETLOG opens the trace and error log when it turns tracing on.
  - *Under IMS TM.* Header, data, and API tracing information are all written to the IMS TM system log.

    The IMS TM system log is always open, but TDSETLOG does a logical OPEN by turning tracing on.
  - *Under MVS.* The information is written to a sequential file.

    Gateway-Library opens the log, but TDSETLOG does a logical OPEN by turning tracing on.

Note This function can be used to send local messages to the trace and error log even when the connection is down.

See also

Related functions

- TDINFLOG on page 123

- TDSETLOG on page 187

APPENDIX A

Gateway-Library Quick Reference

This appendix contains Table A-1, which lists the Gateway-Library functions, shows the arguments used with each, and gives the function's symbolic constants where used.

Table A-1: Gateway-Library function quick reference

Function	Arguments	Symbolic constants
TDACCEPT	(*TDPROC*,	
	RETCODE,	
	IHANDLE,	
	ACCEPT_CONNECTION_NAME,	
	ERROR_SUBCODE);	
TDCONVRT	(*TDPROC*,	
	RETCODE,	
	NUM_DECIMAL_PLACES,	
	SOURCE_TYPE,	
	SOURCE_LENGTH,	
	SOURCE_VARIABLE,	
	RESULT_TYPE,	
	RESULT_LENGTH,	
	RESULT_VARIABLE,	
	OUTLEN);	
	Note *OUTLEN* is optional.	
TDCURPRO	(*TDPROC*,	
	RETCODE,	
	ACTION,	
	CURSOR_DESC);	

Function	Arguments	Symbolic constants
TDESCRIB	(*TDPROC*,	
	RETCODE,	
	COLUMN_NUMBER,	
	HOST_VARIABLE_TYPE,	
	HOST_VARIABLE_MAXLEN,	
	HOST_VARIABLE_NAME,	
	NULL_INDICATOR_VARIABLE,	
	NULLS_ALLOWED,	TDS_TRUE TDS_FALSE
	COLUMN_TYPE,	
	COLUMN_MAXLEN,	
	COLUMN_NAME,	
	COLUMN_NAME_LENGTH);	
TDFREE	(*TDPROC*,	
	RETCODE);	
TDGETREQ	(*TDPROC*,	
	RETCODE,	
	WAIT_OPTION,	TDS_TRUE TDS_FALSE
	REQUEST_TYPE,	TDS_LANGUAGE_EVENT TDS_RPC_EVENT TDS_DYNAMIC_EVENT TDS_CURSOR_EVENT
	TRAN_NAME);	
TDGETSOI	(*TDPROC*,	
	RETCODE,	
	OBJECT_TYPE	TDS_OBJECT_COL TDS_OBJECT_PARM
	OBJECT_NUMBER	
	STRIP_SOSI);	TDS_STRIP_SOSI TDS_BLANK_SOSI

Function	Arguments	Symbolic constants
TDGETUSR	(*TDPROC*,	
	RETCODE,	
	ACCESS_CODE,	
	USER_ID,	
	PASSWORD,	
	SERVER_NAME,	
	CLIENT_CHARSET,	
	NATIONAL_LANGUAGE,	
	SERVER_CHARSET,	
	SERVER_DBCS,	
	APPNAME_ID);	
TDINFACT	(*IHANDLE*,	
	RETCODE,	
	ACCOUNTING_FLAG,	TDS_TRUE TDS_FALSE
	ACCOUNTING_FILENAME,	
	MAXNUM_ACCT_RECORDS);	
TDINFBCD	(*TDPROC*,	
	RETCODE,	
	OBJECT_TYPE,	TDS_OBJECT_COL TDS_OBJECT_PARM
	OBJECT_NUMBER,	
	BCD_LENGTH,	
	BCD_NUMBER_DECIMAL_PLACES);	

Function	Arguments	Symbolic constants
TDINFLOG	(*IHANDLE*,	
	RETCODE,	
	GLOBAL_TRACE_FLAG,	TDS_NO_TRACING TDS_TRACE_ALL_RPCS TDS_TRACE_SPECIFIC_RPCS TDS_TRACE_ERRORS_ONLY
	API_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TDS_HEADER_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TDS_DATA_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TRACE_ID,	
	TRACE_FILENAME,	
	MAXNUM_TRACE_RECORDS);	
TDINFPGM	(*TDPROC*,	
	RETCODE,	
	TDS_VERSION,	TDS_VERSION2_3 TDS_VERSION3_4 TDS_VERSION4_0 TDS_VERSION4_2 TDS_VERSION4_6 TDS_VERSION4_8 TDS_VERSION4_9 TDS_VERSION5_0
	LONGVAR_TRUNC_FLAG,	TDS_TRUE TDS_FALSE
	ROW_LIMIT,	
	REMOTE_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	USER_CORRELATOR,	
	DB2GW_OPTIONS,	
	DB2GW_PID,	
	REQUEST_TYPE);	TDS_LANGUAGE_EVENT TDS_RPC_EVENT TDS_CURSOR_EVENT TDS_DYNAMIC_EVENT

Function	Arguments	Symbolic constants
TDINFPRM	(*TDPROC*,	
	RETCODE,	
	PARM_ID,	
	DATATYPE,	
	ACTUAL_DATA_LENGTH,	
	MAX_DATA_LENGTH,	
	PARM_STATUS,	For TDS 4.6: TDS_INPUT_VALUE TDS_RETURN_VALUE For TDS 5.0: TDS_INPUT_VALUE_NULLABLE TDS_RETURN_VALUE_NULLABLE
	PARM_NAME,	
	PARM_NAME_LENGTH,	
	USER_DATATYPE);	
TDINFRPC	(*TDPROC*,	
	RETCODE,	
	REQUEST_TYPE,	TDS_LANGUAGE_EVENT TDS_RPC_EVENT TDS_CURSOR_EVENT TDS_DYNAMIC_EVENT
	REC_NAME,	
	COMM_STATE);	TDS_RESET TDS_SEND TDS_RECEIVE
TDINFSPT	(*IHANDLE*,	
	RETCODE,	
	TRACE_STATUS,	TDS_TRUE TDS_FALSE
	TRACE_OPTION,	TDS_SPT_API_TRACE TDS_SPT_ERRLOG TDS_SPT_TDS_DATA
	TRANSACTION_ID,	
	TRANSACTION_ID_LENGTH);	
TDINFUDT	(*TDPROC*,	
	RETCODE,	
	COLUMN_NUMBER,	
	USER_DATATYPE);	

Function	Arguments	Symbolic constants
TDINIT	*For CICS:* (*DFHEIBLK*, *For IMS TM:* (*IO_PCB*,	
	RETCODE,	
	IHANDLE);	
TDLOCPRM	(*TDPROC*,	
	PARM_ID,	
	PARM_NAME,	
	PARM_NAME_LENGTH);	
TDLSTSPT	(*IHANDLE*,	
	RETCODE,	
	TRACE_TABLE_LIST);	
TDNUMPRM	(*TDPROC*,	
	NUMBER_OF_PARMS);	
TDRCVPRM	(*TDPROC*,	
	RETCODE,	
	PARM_ID,	
	HOST_VARIABLE,	
	HOST_VARIABLE_TYPE,	
	MAX_DATA_LENGTH,	
	ACTUAL_DATA_LENGTH);	
TDRCVSQL	(*TDPROC*,	
	RETCODE,	
	HOST_VARIABLE,	
	MAX_VAR_LENGTH,	
	ACTUAL_STRING_LENGTH);	
TDRESULT	(*TDPROC*,	
	RETCODE);	
TDSETACT	(*IHANDLE*,	
	RETCODE,	
	ACCOUNTING_FLAG,	TDS_TRUE TDS_FALSE
	ACCOUNTING_FILENAME,	
	MAXNUM_ACCT_RECORDS);	

Function	Arguments	Symbolic constants
TDSETBCD	(*TDPROC*,	
	RETCODE,	
	OBJECT_TYPE,	TDS_OBJECT_COL TDS_OBJECT_PARM
	OBJECT_NUMBER,	
	BCD_LENGTH,	
	BCD_NUMBER_DECIMAL_PLACES);	
TDSETLEN	(*TDPROC*,	
	RETCODE,	
	COLUMN_NUMBER,	
	NEW_COLUMN_LENGTH);	
TDSETLOG	(*IHANDLE*,	
	RETCODE,	
	GLOBAL_TRACE_FLAG,	TDS_NO_TRACING TDS_TRACE_ALL_RPCS TDS_TRACE_SPECIFIC_RPCS TDS_TRACE_ERRORS_ONLY
	API_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TDS_HEADER_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TDS_DATA_TRACE_FLAG,	TDS_TRUE TDS_FALSE
	TRACE_ID	
	TRACE_FILENAME	
	MAXNUM_TRACE_RECORDS);	
TDSETPRM	(*TDPROC*,	
	RETCODE,	
	PARM_ID,	
	HOST_VARIABLE_TYPE,	
	HOST_VARIABLE_LENGTH,	
	HOST_VARIABLE,	
	USER_DATATYPE);	

Function	Arguments	Symbolic constants
TDSETPT	(*IHANDLE*,	
	RETCODE,	
	PROG_TYPE,	MPP BMP CONV EXPL
	SPA,	
	RESERVED1,	
	RESERVED2);	
TDSETSOI	(*TDPROC*,	
	RETCODE,	
	OBJECT_TYPE,	TDS_OBJECT_COLUMN TDS_OBJECT_PARAMETER
	OBJECT_NUMBER,	
	STRIP_SOSI);	TDS_STRIP_SOSI TDS_BLANK_SOSI
TDSETSPT	(*IHANDLE*,	
	RETCODE,	
	TRACE_STATUS,	TDS_TRUE TDS_FALSE
	TRACE_OPTIONS,	TDS_SPT_API_TRACE TDS_SPT_ERRLOG TDS_SPT_TDS_DATA
	TRANSACTION_ID,	
	TRANSACTION_ID_LENGTH);	
TDSETUDT	(*TDPROC*,	
	RETCODE,	
	COLUMN_NUMBER,	
	USER_DATATYPE);	

Function	Arguments	Symbolic constants
TDSNDDON	(*TDPROC*,	
	RETCODE,	
	STATUS,	TDS_DONE_FINAL TDS_DONE_CONTINUE TDS_DONE_ERROR TDS_DONE_COUNT
	ROW_COUNT,	
	RETURN_STATUS_NUMBER,	
	CONN_OPTIONS);	TDS_ENDREPLY TDS-ENDRPC TDS-FLUSH
TDSNDMSG	(*TDPROC*,	
	RETCODE,	
	MESSAGE_TYPE,	TDS_INFO_MSG TDS_ERROR_MSG
	MESSAGE_NUMBER,	
	SEVERITY,	TDS_INFO_SEV TDS_ERROR_SEV
	ERROR_STATE,	
	LINE_ID,	
	TRANSACTION_ID,	
	TRANSACTION_ID_LENGTH,	
	MESSAGE_TEXT,	
	MESSAGE_LENGTH);	
TDSNDROW	(*TDPROC*,	
	RETCODE);	
TDSQLLEN	(*TDPROC*,	
	SQL_LENGTH);	

Function	Arguments	Symbolic constants
TDSTATUS	(*TDPROC*,	
	RETCODE,	
	RETURN_STATUS_NUMBER,	
	DONE_STATUS,	TDS_DONE_FINAL TDS_DONE_CONTINUE TDS_DONE_ERROR TDS_DONE_COUNT
	DONE_COUNT,	
	COMM_STATE,	TDS_RESET TDS_SEND TDS_RECEIVE
	COMM_RETCODE,	
	COMM_ERROR_SUBCODE);	
TDTERM	(*IHANDLE*,	
	RETCODE);	
TDYNAMIC	*TDPROC*,	
	RETCODE	TDS_DYN_TYPE TDS_DYN_IDLEN TDS_DYN_ID TDS_DYN_STMTLEN TDS_DYN_STMT
	CMD,	
	ITEM,	
	HOST_VARIABLE,	
	HOST_VAR_LENGTH,	
	ACTUAL_DATA_LENGTH	
TDWRTLOG	(*TDPROC*,	
	RETCODE,	
	DATETIME_FLAG,	TDS_TRUE TDS_FALSE
	MESSAGE,	
	MESSAGE_LENGTH);	

APPENDIX B

Sample RPC Application for CICS

This appendix contains a sample host server application program that processes a client RPC from the Open Client DB-Library program syr1.c. The server program listed here is included on the Open ServerConnect package; syr1.c is included with TRS.

The purpose of this sample program is to demonstrate the use of Gateway-Library functions, particularly those designed to handle remote procedure calls from a client. In some cases, one Gateway-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 ServerConnect package. It references a table, SYBASE.SAMPLETB, which you create from the SPUFI input file *SYGWSDAT* provided with this product.

This program demonstrates the use of the following Gateway-Library functions listed in Table B-1.

Table B-1: List of functions used in SYCPSAR1

Name	Action
TDACCEPT	Accept a client request.
TDCONVRT	Convert data from host datatype to DB-Library datatype.
TDESCRIB	Describe a column.
TDFREE	Free up the TDPROC structure for the connection.
TDGETUSR	Get user login information from the client.
TDINFBCD	Get BCD information for a described column.
TDINFPRM	Get information about one RPC parameter.
TDINFUDT	Get the user-defined datatype of a column.
TDINIT	Initialize the Gateway-Library environment.
TDLOCPRM	Return ID of one RPC parameter based on name.
TDNUMPRM	Get total number of RPC parameters.
TDRCVPRM	Receive RPC parameter from client program.
TDRESULT	Describe next communication from client.

Name	Action
TDSETBCD	Set scaling for a described column.
TDSETLEN	Set the length for a described column.
TDSETPRM	Set one return parameter.
TDSETUDT	Set a column's user datatype.
TDSNDDON	Send results-completion to client.
TDSNDMSG	Send message to client.
TDSNDROW	Send row to client.
TDSTATUS	Get status information.

Sample program SYCPSAR1

This program accesses the sample DB2 table, SYBASE.SAMPLETB and selects columns from all rows with a department number that matches the number supplied in a passed parameter. It returns the selected rows to the client. The number of rows is returned in a return parameter.

After each row is sent, this program examines the TDSNDROW return code. If it receives a cancel request, it stops sending rows.

If the program completes successfully, it sends a confirmation message to the client; otherwise, it sends an error message.

```
SYSAMP1: PROC OPTIONS(MAIN REENTRANT);
 /*        @(#) sycpsar1.pli 11.1 2/17/95     */
 /****** SYCPSAR1 - RPC REQUEST APPLICATION - PL/I - CICS ************/
 /*                                                                  */
 /*  TRANID:         SYR1                                            */
 /*  PROGRAM:        SYCPSAR1                                        */
 /*  PLAN NAME:      SYR1PLAN                                        */
 /*  FILES:          none                                            */
 /*  TABLES:         SYBASE.SAMPLETB                                 */
 /*                                                                  */
 /*  This program is executed via a client RPC request from sample   */
 /*  dblib program 'SYR1'.  The purpose of the program is primarily  */
 /*  to demonstrate Server Library calls, especially those which     */
 /*  would be used in a server application designed to handle         */
 /*  RPC requests.                                                   */
 /*                                                                  */
 /*  Server Library calls:                                           */
 /*    TDACCEPT      accept request from client                      */
```

```
/*    TDCONVRT      convert data from host to DBlib datatype        */
/*    TDESCRIB      describe a column                               */
/*    TDFREE        free TDPROC structure                           */
/*    TDGETUSR      get user login information from the client      */
/*    TDINFBCD      get BCD information for a described column      */
/*    TDINFPRM      get information about one rpc parameter         */
/*    TDINFUDT      get user column datatype                        */
/*    TDINIT        establish environment                           */
/*    TDLOCPRM      return id of one rpc parameter based on name    */
/*    TDNUMPRM      get total nr of rpc parameters                  */
/*    TDRCVPRM      retrieve rpc parameter from client              */
/*    TDRESULT      describe next communication                     */
/*    TDSETBCD      set scaling for a described column              */
/*    TDSETLEN      set length of a described column                */
/*    TDSETPRM      set return parameter                            */
/*    TDSETUDT      set user column datatype                        */
/*    TDSNDDON      send results-completion to client               */
/*    TDSNDMSG      send message to client                          */
/*    TDSNDROW      send row to client                              */
/*    TDSTATUS      get status information                          */
/*                                                                  */
/*                                                                  */
/*  The program selects columns from the DB2 sample table           */
/*  SYBASE.SAMPLETB of all rows with a department number equal      */
/*  to that supplied in a passed parameter.                         */
/*                                                                  */
/*  The number of rows is returned in a return parameter.           */
/*                                                                  */
/*  After each row is sent, TDSNDROW's return code is examined.     */
/*  If a cancel request was received, then no more rows are sent.   */
/*                                                                  */
/*  A confirmation message is sent to the client if all is          */
/*  well, otherwise an error message is sent.                       */
/*                                                                  */
/*  CHANGE ACTIVITY:                                                */
/*     06/90   - Created, MPM                                       */
/*     10/93   - Added INCLUDE for table definition, TC             */
/*                                                                  */
/******************************************************************/

/*-----------------------------------------------------------------*/
/*        DB2 SQLCA                                                */
/*-----------------------------------------------------------------*/
        EXEC SQL INCLUDE SQLCA;

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

```
/*        TABLE DEFINITION FOR SYBASE.SAMPLETB                    */
/*----------------------------------------------------------------*/
        EXEC SQL INCLUDE SYCPSMPT;

/******************************************************************/
/*        SERVER LIBRARY PL/I COPY BOOK                           */
/******************************************************************/
        %INCLUDE SYGWPLI;


/*----------------------------------------------------------------*/
/*        SERVER LIB ROUTINES DECLARATIONS                        */
/*----------------------------------------------------------------*/
        DCL
            TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
            TDCONVRT ENTRY OPTIONS(INTER ASSEMBLER),
            TDESCRIB ENTRY OPTIONS(INTER ASSEMBLER),
            TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
            TDGETUSR ENTRY OPTIONS(INTER ASSEMBLER),
            TDINFBCD ENTRY OPTIONS(INTER ASSEMBLER),
            TDINFPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDINFUDT ENTRY OPTIONS(INTER ASSEMBLER),
            TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
            TDLOCPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDNUMPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDRCVPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDRESULT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETBCD ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETLEN ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETUDT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDROW ENTRY OPTIONS(INTER ASSEMBLER),
            TDSTATUS ENTRY OPTIONS(INTER ASSEMBLER);

/*----------------------------------------------------------------*/
/*        BUILT IN FUNCTIONS DECLARATIONS                         */
/*----------------------------------------------------------------*/
        DCL
            ADDR      BUILTIN,
            INDEX     BUILTIN,
            LENGTH    BUILTIN,
            NULL      BUILTIN,
            STG       BUILTIN,
            STRING    BUILTIN;
```

```
/*-------------------------------------------------------------------*/
/*        WORK AREAS                                                 */
/*-------------------------------------------------------------------*/
        DCL
            01  GW_LIB_MISC_FIELDS,
                05  GWL_PROC                PTR,
                05  GWL_INIT_HANDLE         PTR,
                05  GWL_RC                  FIXED BIN(31),
                05  GWL_INFPRM_ID           FIXED BIN(31),
                05  GWL_INFPRM_TYPE         FIXED BIN(31),
                05  GWL_INFPRM_DATA_L       FIXED BIN(31),
                05  GWL_INFPRM_MAX_DATA_L   FIXED BIN(31),
                05  GWL_INFPRM_STATUS       FIXED BIN(31),
                05  GWL_INFPRM_NAME         CHAR(30),
                05  GWL_INFPRM_NAME_L       FIXED BIN(31),
                05  GWL_INFPRM_USER_DATA    FIXED BIN(31),
                05  GWL_INFUDT_USER_TYPE    FIXED BIN(31),
                05  GWL_STATUS_NR           FIXED BIN(31),
                05  GWL_STATUS_DONE         FIXED BIN(31),
                05  GWL_STATUS_COUNT        FIXED BIN(31),
                05  GWL_STATUS_COMM         FIXED BIN(31),
                05  GWL_STATUS_RETURN_CODE  FIXED BIN(31),
                05  GWL_STATUS_SUBCODE      FIXED BIN(31),
                05  GWL_NUMPRM_PARMS        FIXED BIN(31),
                05  GWL_RCVPRM_DATA_L       FIXED BIN(31),
                05  GWL_SETPRM_ID           FIXED BIN(31),
                05  GWL_SETPRM_TYPE         FIXED BIN(31),
                05  GWL_SETPRM_DATA_L       FIXED BIN(31),
                05  GWL_SETPRM_USER_DATA    FIXED BIN(31),
                05  GWL_CONVRT_SCALE        FIXED BIN(31) INIT(02),
                05  GWL_SETBCD_SCALE        FIXED BIN(31) INIT(0),
                05  GWL_INFBCD_LENGTH       FIXED BIN(31),
                05  GWL_INFBCD_SCALE        FIXED BIN(31);

        DCL
            01  PARM_FIELDS,
                05  PARM_DEPT             CHAR(3) VAR,
                05  PARM_RETURN_ROWS      FIXED BIN(31) INIT(0);

        DCL
            01  SNA_FIELDS,
                05  SNA_SUBC              FIXED BIN(31),
                05  SNA_CONNECTION_NAME   CHAR(08) INIT(' ');

        DCL
```

```
    01  EMPLOYEE_FIELDS,
        05  EMPLOYEE_FNM          CHAR(12) VAR,
        05  EMPLOYEE_LNM          CHAR(15) VAR,
        05  EMPLOYEE_ED           FIXED BIN(15),
        05  EMPLOYEE_JC           FIXED DEC(3),
        05  EMPLOYEE_SAL          FIXED DEC(8,2);

DCL
    01  COLUMN_NAME_FIELDS,
        05  CN_FNM                CHAR(10) INIT('FIRST_NAME'),
        05  CN_LNM                CHAR(09) INIT('LAST_NAME'),
        05  CN_ED                 CHAR(09) INIT('EDUCATION'),
        05  CN_JC                 CHAR(07) INIT('JOBCODE'),
        05  CN_SAL                CHAR(06) INIT('SALARY');

DCL
    01  DESCRIBE_BIND_FIELDS,
        05  DB_HOST_TYPE          FIXED BIN(31),
        05  DB_CLIENT_TYPE        FIXED BIN(31),
        05  DB_DESCRIBE_HV_PTR    PTR,
        05  DB_COLUMN_NAME_HV_PTR PTR,
        05  DB_NULL_INDICATOR     FIXED BIN(15) INIT(0);

DCL
    01  TDGETUSR_FIELDS,
        05  GU_ACCESS_CODE        CHAR(32),
        05  GU_USER_ID            CHAR(32),
        05  GU_PASSWORD           CHAR(32),
        05  GU_SERVER_NAME        CHAR(32),
        05  GU_CLIENT_CHARSET     CHAR(32),
        05  GU_NATIONAL_LANG      CHAR(32),
        05  GU_SERVER_CHARSET     CHAR(32),
        05  GU_SERVER_DBCS        CHAR(32),
        05  GU_APP_ID             CHAR(32);

DCL
    01  DB_DESCRIBE_HV
            BASED(DB_DESCRIBE_HV_PTR);

DCL
    01  DB_COLUMN_NAME_HV
            BASED(DB_COLUMN_NAME_HV_PTR);

DCL
    01  COUNTER_FIELDS,
        05  CTR_COLUMN            FIXED BIN(31) INIT(0);
```

```
DCL
    01  WORK_FIELDS,
        05  WRKLEN1                FIXED BIN(31),
        05  WRKLEN2                FIXED BIN(31),
        05  WRK_DONE_STATUS        FIXED BIN(31),
        05  WRK_EMPLOYEE_SAL       CHAR(08),
        05  WRK_BLANKS_SS          FIXED BIN(31);

DCL
    01  MESSAGE_FIELDS,
        05  MSG_TYPE               FIXED BIN(31),
        05  MSG_SEVERITY           FIXED BIN(31),
        05  MSG_SEVERITY_OK        FIXED BIN(31) INIT(9),
        05  MSG_SEVERITY_ERROR     FIXED BIN(31) INIT(11),
        05  MSG_NR                 FIXED BIN(31),
        05  MSG_NR_OK              FIXED BIN(31) INIT(1),
        05  MSG_NR_ERROR           FIXED BIN(31) INIT(2),
        05  MSG_RPC                CHAR(04)      INIT('SYR1'),
        05  MSG_RPC_L              FIXED BIN(31)
            INIT(STG(MSG_RPC)),
        05  MSG_TEXT               CHAR(100),
        05  MSG_TEXT_L             FIXED BIN(31),
        05  MSG_NOT_RPC            CHAR(30) INIT
            ('SYR1 not begun via rpc request'),
        05  MSG_NOT_AUTH           CHAR(19) INIT
            ('User not authorized'),
        05  MSG_WRONG_NR_PARMS     CHAR(30) INIT
            ('Number of parameters was not 2'),
        05  MSG_NOT_RETURN_PARM    CHAR(42) INIT
            ('First parameter must be a RETURN parameter'),
        05  MSG_NOT_CHAR_PARM      CHAR(41) INIT
            ('Second parameter must be a CHARACTER type'),
        05  MSG_BAD_CURSOR         CHAR(27) INIT
            ('ERROR - can not open cursor'),
        05  MSG_BAD_FETCH          CHAR(24) INIT
            ('ERROR - fetch row failed'),
        05  MSG_SQL_ERROR,
            10  FILLER1            CHAR(10) INIT
                ('Sqlcode = '),
            10  MSG_SQL_ERROR_C    PIC '---9',
            10  FILLER2            CHAR(16) INIT
                (', Error Tokens: '),
            10  MSG_SQL_ERROR_K    CHAR(70),
        05  MSG_SQL_ERROR_SS       FIXED BIN(15);
```

```
        DCL
            01  MSG_SQLERRM                CHAR(70);

        DCL
            01  MSG_SQLERRM_CHARS(70)      DEFINED MSG_SQLERRM
                                            CHAR(01);

        DCL
            01  CICS_FIELDS,
                05  CICS_RESPONSE          FIXED BIN(31);

        DCL
             01  SWITCHES,
                 05  ALL_DONE              BIT(01) INIT('0'B),
                 05  ALL_DONE_YES          BIT(01) INIT('1'B),
                 05  SEND_DONE             BIT(01) INIT('1'B),
                 05  SEND_DONE_ERROR       BIT(01) INIT('0'B),
                 05  SEND_DONE_OK          BIT(01) INIT('1'B);

/*-------------------------------------------------------------------*/
/*        DECLARE CURSOR                                             */
/*-------------------------------------------------------------------*/
          EXEC SQL
                DECLARE  ECURSOR  CURSOR
                         FOR  SELECT  FIRSTNME,  LASTNAME,
                                      EDUCLVL,   JOBCODE,  SALARY
                              FROM  SYBASE.SAMPLETB
                              WHERE WORKDEPT = :PARM_DEPT;

/*-------------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-------------------------------------------------------------------*/


/*      -------------------------------------------------------------*/
/*      reset db2 error handlers                                     */
/*      -------------------------------------------------------------*/
        EXEC SQL WHENEVER SQLWARNING CONTINUE;
        EXEC SQL WHENEVER SQLERROR   CONTINUE;
        EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/*      -------------------------------------------------------------*/
/*      establish gateway environment                                */
/*      -------------------------------------------------------------*/
        CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);
```

```
/*      ------------------------------------------------------------*/
/*      accept client request                                       */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);

/*      ------------------------------------------------------------*/
/*      TDRESULT to validate kicked off via rpc request             */
/*      ------------------------------------------------------------*/
        CALL TDRESULT (GWL_PROC, GWL_RC);

        IF GWL_RC ^= TDS_PARM_PRESENT THEN
        DO;
            CALL TDRESULT_ERROR;
            GO TO END_PROGRAM;
        END;

/*      ------------------------------------------------------------*/
/*      verify user login information                               */
/*      ------------------------------------------------------------*/
        GU_ACCESS_CODE = 'TOP SECRET';

        CALL TDGETUSR (GWL_PROC, GWL_RC,
                       GU_ACCESS_CODE,
                       GU_USER_ID,
                       GU_PASSWORD,
                       GU_SERVER_NAME,
                       GU_CLIENT_CHARSET,
                       GU_NATIONAL_LANG,
                       GU_SERVER_CHARSET,
                       GU_SERVER_DBCS,
                       GU_APP_ID);

        IF GWL_RC ^= TDS_OK THEN
        DO;
            CALL TDGETUSR_ERROR;
            GO TO END_PROGRAM;
        END;

/*--------------------------------------------------------------------*/
GET_NR_OF_PARMS:
/*--------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      get nr of parms .. better be two                            */
```

```
/*      ------------------------------------------------------------*/
        CALL TDNUMPRM (GWL_PROC, GWL_NUMPRM_PARMS);

        IF GWL_NUMPRM_PARMS ^= 2 THEN
        DO;
            CALL TDNUMPRM_ERROR;
            GO TO END_PROGRAM;
        END;

/*--------------------------------------------------------------------*/
GET_PARMS:
/*--------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      get return parameter information                            */
/*      ------------------------------------------------------------*/
        GWL_INFPRM_ID = 1;
        CALL GET_PARM_INFO;

        (IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE AND
        IF GWL_INFPRM_STATUS ^= TDS_RETURN_VALUE_NULLABLE) THEN
        DO;
            CALL TDINFPRM_NOT_RETURN_PARM_ERROR;
            GO TO END_PROGRAM;
        END;

        GWL_SETPRM_USER_DATA = GWL_INFPRM_USER_DATA;
        GWL_SETPRM_ID        = GWL_INFPRM_ID;
        GWL_SETPRM_DATA_L    = GWL_INFPRM_DATA_L;
        GWL_SETPRM_TYPE      = GWL_INFPRM_TYPE;

/*      ------------------------------------------------------------*/
/*      get department id parameter number from known name          */
/*      ------------------------------------------------------------*/
        GWL_INFPRM_NAME   = '@parm2';
        GWL_INFPRM_NAME_L = 6;

        CALL TDLOCPRM (GWL_PROC,
                       GWL_INFPRM_ID,
                       GWL_INFPRM_NAME,
                       GWL_INFPRM_NAME_L);


/*      ------------------------------------------------------------*/
/*      get department parameter information                        */
/*      ------------------------------------------------------------*/
```

```
        CALL GET_PARM_INFO;

        IF GWL_INFPRM_TYPE ^= TDSVARYCHAR THEN
        DO;
            CALL TDINFPRM_NOT_CHAR_PARM_ERROR;
            GO TO END_PROGRAM;
        END;

/*      ------------------------------------------------------------*/
/*      get department parameter data                               */
/*      ------------------------------------------------------------*/
        CALL TDRCVPRM (GWL_PROC, GWL_RC,
                       GWL_INFPRM_ID,
                       PARM_DEPT,
                       GWL_INFPRM_TYPE,
                       GWL_INFPRM_MAX_DATA_L,
                       GWL_RCVPRM_DATA_L);

/*--------------------------------------------------------------------*/
OPEN_DB2_CURSOR:
/*--------------------------------------------------------------------*/
        EXEC SQL OPEN ECURSOR;

        IF SQLCODE ^= 0 THEN
        DO;
           CALL OPEN_ERROR;
           GO TO END_PROGRAM;
        END;

/*--------------------------------------------------------------------*/
SETUP_REPLY_COLUMNS:
/*--------------------------------------------------------------------*/
        DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_FNM);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_FNM);
        WRKLEN1               = STG(EMPLOYEE_FNM)-2;
        WRKLEN2               = STG(CN_FNM);
        DB_HOST_TYPE          = TDSVARYCHAR;
        DB_CLIENT_TYPE        = TDSVARYCHAR;
        CALL DESCRIBE_COLUMN;

/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR) */
/*      to DBCHAR.                                                  */
/*      ------------------------------------------------------------*/
        DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_LNM);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_LNM);
```

```
        WRKLEN1               = STG(EMPLOYEE_LNM)-2;
        WRKLEN2               = STG(CN_LNM);
        DB_HOST_TYPE          = TDSVARYCHAR;
        DB_CLIENT_TYPE        = TDSCHAR;
        CALL DESCRIBE_COLUMN;

        DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_ED);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_ED);
        WRKLEN1               = STG(EMPLOYEE_ED);
        WRKLEN2               = STG(CN_ED);
        DB_HOST_TYPE          = TDSINT2;
        DB_CLIENT_TYPE        = TDSINT2;
        CALL DESCRIBE_COLUMN;

/*      ------------------------------------------------------------*/
/*      Get the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
        CALL TDINFUDT (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       GWL_INFUDT_USER_TYPE);

/*      ------------------------------------------------------------*/
/*      Set the user defined datatype of EMPLOYEE_ED column.        */
/*      ------------------------------------------------------------*/
        CALL TDSETUDT (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       GWL_INFUDT_USER_TYPE);

/*      ------------------------------------------------------------*/
/*      Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.    */
/*      ------------------------------------------------------------*/
        DB_DESCRIBE_HV_PTR    = ADDR(EMPLOYEE_JC);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_JC);
        WRKLEN1               = STG(EMPLOYEE_JC);
        WRKLEN2               = STG(CN_JC);
        DB_HOST_TYPE          = TDSDECIMAL;
        DB_CLIENT_TYPE        = TDSFLT8;
        CALL DESCRIBE_COLUMN;

/*      ------------------------------------------------------------*/
/*      We must inform the Server Library how many decimal places   */
/*      are in the EMPLOYEE_JC column.                              */
/*      ------------------------------------------------------------*/
        CALL TDSETBCD (GWL_PROC, GWL_RC,
                       TDS_OBJECT_COL,
                       CTR_COLUMN,
```

```
                        TDS_DEFAULT_LENGTH,
                        GWL_SETBCD_SCALE);

/*      ------------------------------------------------------------*/
/*      Demonstrate getting decimal column information.             */
/*      ------------------------------------------------------------*/
        CALL TDINFBCD (GWL_PROC, GWL_RC,
                        TDS_OBJECT_COL,
                        CTR_COLUMN,
                        GWL_INFBCD_LENGTH,
                        GWL_INFBCD_SCALE);



/*      ------------------------------------------------------------*/
/*      Here we intend to use TDCONVRT to convert from TDSDECIMAL to*/
/*      TDSMONEY, so we point TDESCRIB to the output of TDCONVRT,   */
/*      rather than the original input.                             */
/*      ------------------------------------------------------------*/
        DB_DESCRIBE_HV_PTR     = ADDR(WRK_EMPLOYEE_SAL);
        DB_COLUMN_NAME_HV_PTR = ADDR(CN_SAL);
        WRKLEN1                = STG(WRK_EMPLOYEE_SAL);
        WRKLEN2                = STG(CN_SAL);
        DB_HOST_TYPE           = TDSMONEY;
        DB_CLIENT_TYPE         = TDSMONEY;
        CALL DESCRIBE_COLUMN;

/*--------------------------------------------------------------------*/
SEND_ROWS:
/*--------------------------------------------------------------------*/
        DO WHILE(^ ALL_DONE);
           CALL FETCH_AND_SEND_ROWS;
        END;

/*--------------------------------------------------------------------*/
END_OF_QUERY:
/*--------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      close cursor                                                */
/*      ------------------------------------------------------------*/
        EXEC SQL CLOSE ECURSOR;

/*      ------------------------------------------------------------*/
/*      update return parameter with nr of rows fetched             */
/*      ------------------------------------------------------------*/
```

```
        CALL TDSETPRM (GWL_PROC, GWL_RC,
                       GWL_SETPRM_ID,
                       GWL_SETPRM_TYPE,
                       GWL_SETPRM_DATA_L,

                       PARM_RETURN_ROWS,
                       GWL_SETPRM_USER_DATA);

        GO TO END_PROGRAM;

/*--------------------------------------------------------------------*/
FETCH_AND_SEND_ROWS: PROC;
/*--------------------------------------------------------------------*/
        EXEC SQL FETCH ECURSOR INTO :EMPLOYEE_FIELDS;

        IF SQLCODE = 0 THEN
        DO;
/*          ------------------------------------------------------*/
/*          Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.   */
/*          ------------------------------------------------------*/
            WRKLEN1 = STG(EMPLOYEE_SAL);
            WRKLEN2 = STG(WRK_EMPLOYEE_SAL);

            CALL TDCONVRT (GWL_PROC, GWL_RC,
                           GWL_CONVRT_SCALE,
                           TDSDECIMAL,
                           WRKLEN1,
                           EMPLOYEE_SAL,
                           TDSMONEY,
                           WRKLEN2,
                           WRK_EMPLOYEE_SAL);

/*          ------------------------------------------------------*/
/*          Do not send trailing blanks of EMPLOYEE_LNM             */
/*          ------------------------------------------------------*/
            WRKLEN1 = LENGTH(EMPLOYEE_LNM);
            CTR_COLUMN = 2;

            WRK_BLANKS_SS = 1;
LOOP:       DO WHILE(WRK_BLANKS_SS <= WRKLEN1);
              IF SUBSTR(EMPLOYEE_LNM, WRK_BLANKS_SS, 1) = ' ' THEN DO;
                LEAVE LOOP;
              END;

              WRK_BLANKS_SS = WRK_BLANKS_SS + 1;
            END LOOP;
```

```
            IF (WRK_BLANKS_SS <= WRKLEN1) THEN DO;
              CALL TDSETLEN (GWL_PROC, GWL_RC,
                             CTR_COLUMN,
                             WRK_BLANKS_SS - 1);
            END;

/*          ------------------------------------------------------*/
/*          send a row to the client                              */
/*          ------------------------------------------------------*/
            CALL TDSNDROW (GWL_PROC, GWL_RC);
            PARM_RETURN_ROWS = PARM_RETURN_ROWS + 1;

            IF GWL_RC = TDS_CANCEL_RECEIVED THEN
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;
        END;

        ELSE IF SQLCODE = +100 THEN
            DO;
               ALL_DONE = ALL_DONE_YES;
            END;

        ELSE IF SQLCODE < 0 THEN
        DO;
            ALL_DONE = ALL_DONE_YES;
            CALL FETCH_ERROR;
        END;

        RETURN;

END FETCH_AND_SEND_ROWS;

/*--------------------------------------------------------------------*/
GET_PARM_INFO: PROC;
/*--------------------------------------------------------------------*/
        CALL TDINFPRM (GWL_PROC, GWL_RC,
                       GWL_INFPRM_ID,
                       GWL_INFPRM_TYPE,
                       GWL_INFPRM_DATA_L,
                       GWL_INFPRM_MAX_DATA_L,
                       GWL_INFPRM_STATUS,
                       GWL_INFPRM_NAME,
                       GWL_INFPRM_NAME_L,
                       GWL_INFPRM_USER_DATA);
```

```
        RETURN;

END GET_PARM_INFO;

/*--------------------------------------------------------------------*/
DESCRIBE_COLUMN: PROC;
/*--------------------------------------------------------------------*/
        CTR_COLUMN = CTR_COLUMN +1;

        CALL TDESCRIB (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       DB_HOST_TYPE,
                       WRKLEN1,
                       DB_DESCRIBE_HV,
                       DB_NULL_INDICATOR,
                       TDS_FALSE,
                       DB_CLIENT_TYPE,
                       WRKLEN1,
                       DB_COLUMN_NAME_HV,
                       WRKLEN2);

        RETURN;

END DESCRIBE_COLUMN;

/*--------------------------------------------------------------------*/
TDGETUSR_ERROR: PROC;
/*--------------------------------------------------------------------*/
        MSG_TEXT   = MSG_NOT_AUTH;
        MSG_TEXT_L = STG(MSG_NOT_AUTH);
        CALL SEND_ERROR_MESSAGE;

        RETURN;

END TDGETUSR_ERROR;

/*--------------------------------------------------------------------*/
TDRESULT_ERROR: PROC;
/*--------------------------------------------------------------------*/
        MSG_TEXT   = MSG_NOT_RPC;
        MSG_TEXT_L = STG(MSG_NOT_RPC);
        CALL SEND_ERROR_MESSAGE;

        RETURN;
```

```
END TDRESULT_ERROR;

/*-------------------------------------------------------------------*/
TDNUMPRM_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_TEXT   = MSG_WRONG_NR_PARMS;
        MSG_TEXT_L = STG(MSG_WRONG_NR_PARMS);
        CALL SEND_ERROR_MESSAGE;

        RETURN;

END TDNUMPRM_ERROR;

/*-------------------------------------------------------------------*/
TDINFPRM_NOT_RETURN_PARM_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_TEXT   = MSG_NOT_RETURN_PARM;
        MSG_TEXT_L = STG(MSG_NOT_RETURN_PARM);
        CALL SEND_ERROR_MESSAGE;

        RETURN;

END TDINFPRM_NOT_RETURN_PARM_ERROR;

/*-------------------------------------------------------------------*/
TDINFPRM_NOT_CHAR_PARM_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_TEXT   = MSG_NOT_CHAR_PARM;
        MSG_TEXT_L = STG(MSG_NOT_CHAR_PARM);
        CALL SEND_ERROR_MESSAGE;

        RETURN;

END TDINFPRM_NOT_CHAR_PARM_ERROR;



/*-------------------------------------------------------------------*/
OPEN_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_TEXT   = MSG_BAD_CURSOR;
        MSG_TEXT_L = STG(MSG_BAD_CURSOR);
        CALL SEND_ERROR_MESSAGE;
        CALL SEND_SQL_ERROR;
```

```
        RETURN;

END OPEN_ERROR;

/*-------------------------------------------------------------------*/
FETCH_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_TEXT   = MSG_BAD_FETCH;
        MSG_TEXT_L = STG(MSG_BAD_FETCH);
        CALL SEND_ERROR_MESSAGE;
        CALL SEND_SQL_ERROR;

        RETURN;

END FETCH_ERROR;

/*-------------------------------------------------------------------*/
SEND_SQL_ERROR: PROC;
/*-------------------------------------------------------------------*/
        MSG_SQL_ERROR_C = SQLCODE;
        MSG_SQLERRM     = SQLERRM;

/*      ------------------------------------------------------------*/
/*      ensure possible non-printables translated to spaces         */
/*      ------------------------------------------------------------*/
        DO MSG_SQL_ERROR_SS = 1 TO LENGTH(SQLERRM);

            IF MSG_SQLERRM_CHARS(MSG_SQL_ERROR_SS) < ' ' |
               MSG_SQLERRM_CHARS(MSG_SQL_ERROR_SS) > '9' THEN
            DO;
               MSG_SQLERRM_CHARS(MSG_SQL_ERROR_SS) = ' ';
            END;
        END;

        MSG_SQL_ERROR_K = MSG_SQLERRM;
        MSG_TEXT        = STRING(MSG_SQL_ERROR);
        MSG_TEXT_L      = STG(MSG_SQL_ERROR);
        CALL SEND_ERROR_MESSAGE;

        RETURN;

END SEND_SQL_ERROR;


/*-------------------------------------------------------------------*/
SEND_ERROR_MESSAGE: PROC;
```

```
/*-------------------------------------------------------------------*/
        SEND_DONE    = SEND_DONE_ERROR;
        MSG_SEVERITY = MSG_SEVERITY_ERROR;
        MSG_NR       = MSG_NR_ERROR;
        MSG_TYPE     = TDS_ERROR_MSG;
        CALL SEND_MESSAGE;

        RETURN;

END SEND_ERROR_MESSAGE;

/*-------------------------------------------------------------------*/
SEND_MESSAGE: PROC;
/*-------------------------------------------------------------------*/

/*      ---------------------------------------------------------*/
/*      ensure we're in right state to send a message             */
/*      ---------------------------------------------------------*/
        CALL TDSTATUS (GWL_PROC, GWL_RC,
                       GWL_STATUS_NR,
                       GWL_STATUS_DONE,
                       GWL_STATUS_COUNT,
                       GWL_STATUS_COMM,
                       GWL_STATUS_RETURN_CODE,
                       GWL_STATUS_SUBCODE);

        IF GWL_RC = TDS_OK THEN
        DO;
            IF GWL_STATUS_COMM = TDS_RECEIVE THEN

            CALL TDSNDMSG (GWL_PROC, GWL_RC,
                           MSG_TYPE, MSG_NR,
                           MSG_SEVERITY,
                           TDS_ZERO,
                           TDS_ZERO,
                           MSG_RPC, MSG_RPC_L,
                           MSG_TEXT, MSG_TEXT_L);
        END;

        RETURN;

END SEND_MESSAGE;



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

```
END_PROGRAM:
/*--------------------------------------------------------------------*/
        IF SEND_DONE = SEND_DONE_OK THEN
            WRK_DONE_STATUS = TDS_DONE_COUNT;

        ELSE
        DO;
            WRK_DONE_STATUS  = TDS_DONE_ERROR;
            PARM_RETURN_ROWS = 0;
        END;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       WRK_DONE_STATUS,
                       PARM_RETURN_ROWS,
                       TDS_ZERO,
                       TDS_ENDRPC);

        CALL TDFREE (GWL_PROC, GWL_RC);
        EXEC CICS RETURN;

END SYSAMP1;
```

APPENDIX C

Sample Language Application for CICS

This appendix contains a sample Open ServerConnect application program that processes a client's SQL language request using the DB2 dynamic SQL facility. This CICS program uses PL/1, DB2, and Gateway-Library.

The client language request can be entered on line, using ISQL or another Sybase or third party front end, or it can be coded in a DB-Library program. A corresponding DB-Library program, syl1.c, is included with TRS. The server program listed here is included on the Open ServerConnect tape.

If the TRS security administrator specifies this program as your language handler, be sure that syl1.c is the Language RPC Name in the Transaction Group associated with all client logins that use this program to process SQL language requests.

If you want to allow a client to execute this program on line, be sure that the TRS specifies SYL1 rather than AMD2 for SQL language requests.

The purpose of this sample program is to demonstrate the use of Gateway-Library functions, particularly those designed to handle client language requests. In some cases, one Gateway-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.

Note You can write language handling programs to handle any incoming text. You are not restricted to SQL text or to any particular host access method.

This program demonstrates the use of the following Gateway-Library functions listed in Table C-1.

Table C-1: List of functions used in SYCPSAL1

Name	Action
TDACCEPT	Accept a client request.
TDFREE	Free up the TDPROC structure for the connection.
TDINFLOG	Return current trace settings for trace log.
TDINFPGM	Return information about current program.
TDINIT	Initialize the Gateway-Library environment.
TDRCVSQL	Receive a SQL command string from client.
TDRESULT	Describe next communication from client.
TDSETSPT	Set specific tracing.
TDSNDDON	Send results-completion to client.
TDSNDMSG	Send message to client.
TDSQLLEN	Get length of incoming text.
TDSTATUS	Get status information.

Sample program SYCPSAL1

The following program accepts all valid dynamic SQL requests except select commands. delete requests must have a where clause, or they will be rejected. Upon successful completion, this program sends a confirmation message to the client; otherwise it sends an error message.

```
SYSAMP1: PROC OPTIONS(MAIN REENTRANT);
/*        @(#) sycpsal1.pli 1.1 3/17/98     */
/****** SYCPSAL1 - LANGUAGE REQUEST APPLICATION - PL/I - CICS *******/
/*                                                                  */
/*  TRANID:         SYL1                                            */
/*  PROGRAM:        SYCPSAL1                                        */
/*  PLAN NAME:      SYL1PLAN                                        */
/*  FILES:          none                                            */
/*  TABLES:         adhoc                                           */
/*                                                                  */
/*  This program is executed via a client language request          */
/*  from sample dblib program 'SYL1', or by SYBASE's ISQL if        */
/*  installed.  The client program must login to a transaction      */
/*  group with SYL1 as the language handler.                        */
/*                                                                  */
/*  The purpose of the program is primarily to demonstrate Server   */
/*  Library calls, especially those which would be used in a        */
/*  server application designed to handle language requests.        */
```

```
/*                                                                  */
/*  Server Library calls:                                           */
/*   TDACCEPT      accept request from client                       */
/*   TDFREE        free TDPROC structure                            */
/*   TDINFLOG      return trace settings                            */
/*   TDINFPGM      return program information                       */
/*   TDINIT        establish environment                            */
/*   TDRCVSQL      receive language text                            */
/*   TDRESULT      describe next communication                      */
/*   TDSETSPT      set specific tracing                             */
/*   TDSNDDON      send results-completion to client                */
/*   TDSNDMSG      send message to client                           */
/*   TDSQLLEN      get length of incoming text                      */
/*   TDSTATUS      get status information                           */
/*                                                                  */
/*                                                                  */
/*  The program accepts all valid SQL requests other than           */
/*  'SELECT'.  A 'DELETE' must have a WHERE clause, or it is        */
/*  rejected.                                                       */
/*                                                                  */
/*  A confirmation message is sent to the client if all is          */
/*  well, otherwise an error message is sent.                       */
/*                                                                  */
/*  CHANGE ACTIVITY:                                                */
/*     6/90    - Created, MPM                                       */
/*                                                                  */
/******************************************************************/

/*----------------------------------------------------------------*/
/*        DB2 SQLCA                                               */
/*----------------------------------------------------------------*/
       EXEC SQL INCLUDE SQLCA;

/*----------------------------------------------------------------*/
/*        minimum SQLDA                                           */
/*----------------------------------------------------------------*/
       DCL
           01  SQLDA,
               05  SQLDAID              CHAR(8)       INIT(' '),
               05  SQLDABC              FIXED BIN(31) INIT(0),
               05  SQLN                 FIXED BIN(15) INIT(0),
               05  SQLD                 FIXED BIN(15) INIT(0);

/******************************************************************/
/*        SERVER LIBRARY PL/I COPY BOOK                           */
/******************************************************************/
```

```
        %INCLUDE SYGWPLI;

/*--------------------------------------------------------------------*/
/*        SERVER LIB ROUTINES DECLARATIONS                            */
/*--------------------------------------------------------------------*/
        DCL
            TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
            TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
            TDINFLOG ENTRY OPTIONS(INTER ASSEMBLER),
            TDINFPGM ENTRY OPTIONS(INTER ASSEMBLER),
            TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
            TDRCVSQL ENTRY OPTIONS(INTER ASSEMBLER),
            TDRESULT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETSPT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
            TDSQLLEN ENTRY OPTIONS(INTER ASSEMBLER),
            TDSTATUS ENTRY OPTIONS(INTER ASSEMBLER);

/*--------------------------------------------------------------------*/
/*        BUILT IN FUNCTIONS DECLARATIONS                             */
/*--------------------------------------------------------------------*/
        DCL
            ADDR      BUILTIN,
            INDEX     BUILTIN,
            LENGTH    BUILTIN,
            NULL      BUILTIN,
            STG       BUILTIN,
            STRING    BUILTIN,
            SUBSTR    BUILTIN,
            TRANSLATE BUILTIN,
            VERIFY    BUILTIN;

/*--------------------------------------------------------------------*/
/*        WORK AREAS                                                  */
/*--------------------------------------------------------------------*/
        DCL
            01  GW_LIB_MISC_FIELDS,
                05  GWL_PROC                   PTR,
                05  GWL_INIT_HANDLE            PTR,
                05  GWL_RC                     FIXED BIN(31),
                05  GWL_SQLLEN                 FIXED BIN(31),
                05  GWL_STATUS_NR              FIXED BIN(31),
                05  GWL_STATUS_DONE            FIXED BIN(31),
                05  GWL_STATUS_COUNT           FIXED BIN(31),
                05  GWL_STATUS_COMM            FIXED BIN(31),
```

```
        05  GWL_STATUS_RETURN_CODE      FIXED BIN(31),
        05  GWL_STATUS_SUBCODE          FIXED BIN(31),
        05  GWL_INFPGM_TDS_VERSION      FIXED BIN(31),
        05  GWL_INFPGM_LONGVAR          FIXED BIN(31),
        05  GWL_INFPGM_ROW_LIMIT        FIXED BIN(31),
        05  GWL_INFPGM_REMOTE_TRACE     FIXED BIN(31),
        05  GWL_INFPGM_CORRELATOR       FIXED BIN(31),
        05  GWL_INFPGM_DB2GW_OPTION     FIXED BIN(31),
        05  GWL_INFPGM_DB2GW_PID        FIXED BIN(31),
        05  GWL_INFPGM_TYPE_RPC         FIXED BIN(31),
        05  GWL_INFLOG_GLOBAL           FIXED BIN(31),
        05  GWL_INFLOG_API              FIXED BIN(31),
        05  GWL_INFLOG_TDS_HEADER       FIXED BIN(31),
        05  GWL_INFLOG_TDS_DATA         FIXED BIN(31),
        05  GWL_INFLOG_TRACE_ID         FIXED BIN(31),
        05  GWL_INFLOG_FILENAME         CHAR(08),
        05  GWL_INFLOG_TOTAL_RECS       FIXED BIN(31),
        05  GWL_SETSPT_TRACE_LEVEL      FIXED BIN(31),
        05  GWL_SETSPT_RPC_NAME         CHAR(04) INIT('SYL1'),
        05  GWL_SETSPT_RPC_NAME_L       FIXED BIN(31) INIT(4);

DCL
    01  LANGUAGE_FIELDS,
        05  LANG_BUFFER_PTR       PTR,
        05  LANG_MAX_L            FIXED BIN(31),
        05  LANG_ACTUAL_L         FIXED BIN(31),
        05  LANG_TEXT_SS          FIXED BIN(15) INIT(1);

DCL
    01  LANG_BUFFER_DB2           CHAR(1024) VAR;

DCL
    01  LANG_BUFFER_LL_TEXT       BASED(LANG_BUFFER_PTR),


       05  LANG_BUFFER_LL        FIXED BIN(15),
        05  LANG_BUFFER_TEXT      CHAR(1024);

DCL
    01  PARSESQL_BUFFER           BASED(LANG_BUFFER_PTR),
        05  PARSESQL_TEXT_LL      FIXED BIN(15),
        05  PARSESQL_TEXT_CHARS(1024)
                                   CHAR(01);

DCL
    01  SNA_FIELDS,
```

```
        05  SNA_SUBC               FIXED BIN(31),
        05  SNA_CONNECTION_NAME    CHAR(08) INIT(' ');

DCL
    01  PARSE_FIELDS,
        05  PARSE_BYTES            FIXED BIN(31),
        05  PARSE_SS               FIXED BIN(15) INIT(0),
        05  PARSE_ACTION           CHAR(06);

DCL
    01  XLATE_FIELDS               STATIC,
        05  XLATE_LOWER            CHAR(07) INIT('dehlrtw'),
        05  XLATE_UPPER            CHAR(07) INIT('DEHLRTW');

DCL
    01  WORK_FIELDS,
        05  WRK_DONE_STATUS        FIXED BIN(31),
        05  WRK_XLATE_WHERE        CHAR(06);

DCL
    01  MESSAGE_FIELDS,
        05  MSG_TYPE               FIXED BIN(31),
        05  MSG_SEVERITY           FIXED BIN(31),
        05  MSG_SEVERITY_OK        FIXED BIN(31) INIT(9),
        05  MSG_SEVERITY_ERROR     FIXED BIN(31) INIT(11),
        05  MSG_NR                 FIXED BIN(31),
        05  MSG_NR_OK              FIXED BIN(31) INIT(1),
        05  MSG_NR_ERROR           FIXED BIN(31) INIT(2),
        05  MSG_RPC                CHAR(04)      INIT('SYL1'),
        05  MSG_RPC_L              FIXED BIN(31)
            INIT(STG(MSG_RPC)),
        05  MSG_TEXT               CHAR(50),
        05  MSG_TEXT_L             FIXED BIN(31),
        05  MSG_SQL_ERROR,
            10  MSG_SQL_ERROR_T    CHAR(31)
                INIT('Invalid sql request, sqlcode = '),
            10  MSG_SQL_ERROR_C    PIC '---9',
        05  MSG_SELECT             CHAR(24)
                INIT('SQL select not supported'),
        05  MSG_NOT_LANG           CHAR(35)
                INIT('SYL1 not begun via language request'),
        05  MSG_BAD_LEN            CHAR(31)
                INIT('Request has too many characters'),
        05  MSG_NO_WHERE           CHAR(26)
                INIT('Delete has no where clause'),
        05  MSG_OK                 CHAR(22)
```

```
                    INIT('Execute was successful'),
               05  MSG_NOT_OK,
                   10  FILLER1           CHAR(26)
                       INIT('Execute failed, sqlcode = '),
                   10  MSG_NOT_OK_C      PIC '---9',
                   10  FILLER2           CHAR(18)
                       INIT(', ROLLBACK issued.');
        DCL
            01  CICS_FIELDS,
                05  CICS_RESPONSE        FIXED BIN(31);

        DCL
             01  SWITCHES,
                 05  TRACING_SET_SW       BIT(01) INIT('0'B),
                 05  TRACING_RESET        BIT(01) INIT('0'B),
                 05  TRACING_SET          BIT(01) INIT('1'B),
                 05  ALL_DONE             BIT(01) INIT('0'B),
                 05  ALL_DONE_YES         BIT(01) INIT('1'B),
                 05  SEND_DONE            BIT(01) INIT('1'B),
                 05  SEND_DONE_ERROR      BIT(01) INIT('0'B),
                 05  SEND_DONE_OK         BIT(01) INIT('1'B);

/*-------------------------------------------------------------------*/
/*        DECLARE STATEMENT AND CURSOR                               */
/*-------------------------------------------------------------------*/
          EXEC SQL DECLARE S1 STATEMENT;
          EXEC SQL DECLARE C1 CURSOR FOR S1;



/*-------------------------------------------------------------------*/
INITIALIZE_PROGRAM:

/*-------------------------------------------------------------------*/

/*      -------------------------------------------------------------*/
/*      reset db2 error handlers                                     */
/*      -------------------------------------------------------------*/
        EXEC SQL WHENEVER SQLWARNING CONTINUE;
        EXEC SQL WHENEVER SQLERROR   CONTINUE;
        EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/*      -------------------------------------------------------------*/
/*      establish gateway environment                                */
/*      -------------------------------------------------------------*/
        CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);
```

```
/*      ------------------------------------------------------------*/
/*      turn on local tracing if not on globally or locally         */
/*      ------------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_TDS_HEADER,
                       GWL_INFLOG_TDS_DATA,
                       GWL_INFLOG_TRACE_ID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_TOTAL_RECS);

        IF GWL_INFLOG_GLOBAL ^= TDS_TRACE_ALL_RPCS &
           GWL_INFLOG_GLOBAL ^= TDS_TRACE_SPECIFIC_RPCS THEN
        DO;
            TRACING_SET_SW = TRACING_SET;
            CALL LOCAL_TRACING;
        END;

/*      ------------------------------------------------------------*/
/*      accept client request                                       */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);

/*      ------------------------------------------------------------*/
/*      ensure kicked off via language request                      */
/*      (this could be handled more reasonably by TDRESULT)         */
/*      ------------------------------------------------------------*/
        CALL TDINFPGM (GWL_PROC, GWL_RC,
                       GWL_INFPGM_TDS_VERSION,
                       GWL_INFPGM_LONGVAR,
                       GWL_INFPGM_ROW_LIMIT,
                       GWL_INFPGM_REMOTE_TRACE,
                       GWL_INFPGM_CORRELATOR,
                       GWL_INFPGM_DB2GW_OPTION,
                       GWL_INFPGM_DB2GW_PID,
                       GWL_INFPGM_TYPE_RPC);

        IF GWL_INFPGM_TYPE_RPC ^= TDS_START_SQL THEN
        DO;
            MSG_TEXT   = MSG_NOT_LANG;
            MSG_TEXT_L = STG(MSG_NOT_LANG);
            CALL SEND_ERROR_MESSAGE;
```

```
            GO TO END_PROGRAM;
        END;

/*------------------------------------------------------------------*/
READ_IN_SQL_TEXT:
/*------------------------------------------------------------------*/

/*      ------------------------------------------------------------*/
/*      prepare for receive                                         */
/*      ------------------------------------------------------------*/
        CALL TDRESULT (GWL_PROC, GWL_RC);

/*      ------------------------------------------------------------*/
/*      load ptr to redefined sql text                              */
/*      ------------------------------------------------------------*/
        LANG_BUFFER_PTR = ADDR(LANG_BUFFER_DB2);

/*      ------------------------------------------------------------*/
/*      get len of language text, ensure not too big for us         */
/*      (this could be handled without TDSQLLEN by checking         */
/*      LANG_ACTUAL_LEN doesn't exceed LANG_MAX_L in TDRCVSQL call) */
/*      ------------------------------------------------------------*/
        CALL TDSQLLEN (GWL_PROC, GWL_SQLLEN);

        LANG_MAX_L = STG(LANG_BUFFER_TEXT);

        IF GWL_SQLLEN > LANG_MAX_L THEN
        DO;
            MSG_TEXT   = MSG_BAD_LEN;
            MSG_TEXT_L = STG(MSG_BAD_LEN);
            CALL SEND_ERROR_MESSAGE;
            GO TO END_PROGRAM;
        END;

/*      ------------------------------------------------------------*/
/*      get language text                                           */
/*      ------------------------------------------------------------*/
        CALL TDRCVSQL (GWL_PROC, GWL_RC,
                       LANG_BUFFER_TEXT,
                       LANG_MAX_L,
                       LANG_ACTUAL_L);

        LANG_BUFFER_LL = LANG_ACTUAL_L;

/*------------------------------------------------------------------*/
EDIT_TEXT:
```

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

/*      -------------------------------------------------------------*/
/*      ensure line feeds, low-values, etc. translated to blanks     */
/*      -------------------------------------------------------------*/
        DO WHILE (LANG_TEXT_SS <= PARSESQL_TEXT_LL);
            IF  PARSESQL_TEXT_CHARS(LANG_TEXT_SS) < ' ' THEN
            DO;
                PARSESQL_TEXT_CHARS(LANG_TEXT_SS) = ' ';
            END;

            ELSE
            DO;
                IF PARSE_SS = 0 &
                   PARSESQL_TEXT_CHARS(LANG_TEXT_SS) > ' ' THEN
                DO;
                    PARSE_SS    = LANG_TEXT_SS;
                    PARSE_BYTES = PARSESQL_TEXT_LL - LANG_TEXT_SS;
                END;
            END;

            LANG_TEXT_SS = LANG_TEXT_SS +1;
        END;

/*      -------------------------------------------------------------*/
/*      let DB2 edit and tell us if SELECT                           */
/*      -------------------------------------------------------------*/
        EXEC SQL PREPARE S1 INTO SQLDA FROM :LANG_BUFFER_DB2;

        IF SQLD ^= 0 THEN
        DO;
            MSG_TEXT   = MSG_SELECT;
            MSG_TEXT_L = STG(MSG_SELECT);
            CALL SEND_ERROR_MESSAGE;
            GO TO END_PROGRAM;
        END;

        IF SQLCODE < 0 THEN
        DO;
            MSG_SQL_ERROR_C = SQLCODE;
            MSG_TEXT        = STRING(MSG_SQL_ERROR);
            MSG_TEXT_L      = STG(MSG_SQL_ERROR);
            CALL SEND_ERROR_MESSAGE;
            GO TO END_PROGRAM;
        END;
```

```
/*      ------------------------------------------------------------*/
/*      parse and handle special case of DELETE without WHERE clause*/
/*      ------------------------------------------------------------*/
        PARSE_ACTION = TRANSLATE(SUBSTR(LANG_BUFFER_TEXT,
                                        PARSE_SS,
                                        STG(PARSE_ACTION)),
                                 XLATE_UPPER,
                                 XLATE_LOWER);

        IF PARSE_ACTION ^= 'DELETE' THEN
        DO;
            GO TO EXECUTE_SQL;
        END;

        ELSE DO;
/*          ------------------------------------------------------*/
/*          skip past delete to from                              */
/*          ------------------------------------------------------*/
            CALL SKIP_TO_BLANK;
            CALL SKIP_BLANKS;

/*          ------------------------------------------------------*/
/*          skip past from to tableid                             */
/*          ------------------------------------------------------*/
            CALL SKIP_TO_BLANK;
            CALL SKIP_BLANKS;

/*          ------------------------------------------------------*/
/*          skip past tableid to correlator or where              */
/*          ------------------------------------------------------*/
            CALL SKIP_TO_BLANK;
            CALL SKIP_BLANKS;

            IF PARSE_BYTES > STG(WRK_XLATE_WHERE) THEN
            DO;
                WRK_XLATE_WHERE =
                    TRANSLATE(SUBSTR(LANG_BUFFER_TEXT,
                                     PARSE_SS,
                                     STG(WRK_XLATE_WHERE)),
                              XLATE_UPPER,
                              XLATE_LOWER);

                IF WRK_XLATE_WHERE = 'WHERE ' THEN
                DO;
```

```
                    GO TO EXECUTE_SQL;
                END;

/*              ------------------------------------------------------*/
/*              skip correlator                                       */
/*              ------------------------------------------------------*/
                CALL SKIP_TO_BLANK;

                IF PARSE_BYTES > 0 THEN
                DO;
                    CALL SKIP_TO_BLANK;

                    IF PARSE_BYTES > 0 THEN
                    DO;
                        GO TO EXECUTE_SQL;
                    END;
                END;
            END;
        END;

        MSG_TEXT   = MSG_NO_WHERE;
        MSG_TEXT_L = STG(MSG_NO_WHERE);
        CALL SEND_ERROR_MESSAGE;

        GO TO END_PROGRAM;

/*--------------------------------------------------------------------*/
SKIP_TO_BLANK: PROC;
/*--------------------------------------------------------------------*/
        DCL J                              FIXED BIN(15) INIT(0);

        IF PARSE_SS < LANG_BUFFER_LL THEN
        DO;
            J = INDEX(SUBSTR(LANG_BUFFER_TEXT,
                             PARSE_SS,
                             PARSE_BYTES),
                      ' ');
        END;

        IF J = 0 THEN
        DO;
            PARSE_SS    = LANG_BUFFER_LL +1;
            PARSE_BYTES = -1;
        END;

        ELSE DO;
```

```
            PARSE_SS    = PARSE_SS    +J -1;
            PARSE_BYTES = PARSE_BYTES -J +1;
        END;

        RETURN;

END SKIP_TO_BLANK;

/*--------------------------------------------------------------------*/
SKIP_BLANKS: PROC;
/*--------------------------------------------------------------------*/
        DCL J                              FIXED BIN(15) INIT(0);

        IF PARSE_SS < LANG_BUFFER_LL THEN
        DO;
            J = VERIFY(SUBSTR(LANG_BUFFER_TEXT,
                              PARSE_SS,
                              PARSE_BYTES),
                       ' ');
        END;

        IF J = 0 THEN
        DO;
            PARSE_SS    = LANG_BUFFER_LL +1;
            PARSE_BYTES = -1;
        END;

        ELSE DO;
            PARSE_SS    = PARSE_SS    +J -1;
            PARSE_BYTES = PARSE_BYTES -J +1;
        END;

        RETURN;

END SKIP_BLANKS;

/*--------------------------------------------------------------------*/
EXECUTE_SQL:
/*--------------------------------------------------------------------*/
        EXEC SQL EXECUTE S1;

        IF SQLCODE < 0 THEN
        DO;
            CALL CICS_ROLLBACK;
            MSG_NOT_OK_C = SQLCODE;
            MSG_TEXT     = STRING(MSG_NOT_OK);
```

```
            MSG_TEXT_L    = STG(MSG_NOT_OK);
            CALL SEND_ERROR_MESSAGE;
            GO TO END_PROGRAM;
        END;

        MSG_TEXT   = MSG_OK;
        MSG_TEXT_L = STG(MSG_OK);
        CALL SEND_CONFIRM_MESSAGE;
        GO TO END_PROGRAM;

/*--------------------------------------------------------------------*/
SEND_CONFIRM_MESSAGE: PROC;
/*--------------------------------------------------------------------*/
        MSG_SEVERITY = MSG_SEVERITY_OK;
        MSG_NR       = MSG_NR_OK;
        MSG_TYPE     = TDS_INFO_MSG;
        CALL SEND_MESSAGE;

        RETURN;

END SEND_CONFIRM_MESSAGE;

/*--------------------------------------------------------------------*/
SEND_ERROR_MESSAGE: PROC;
/*--------------------------------------------------------------------*/
        SEND_DONE    = SEND_DONE_ERROR;
        MSG_SEVERITY = MSG_SEVERITY_ERROR;
        MSG_NR       = MSG_NR_ERROR;
        MSG_TYPE     = TDS_ERROR_MSG;
        CALL SEND_MESSAGE;

        RETURN;

END SEND_ERROR_MESSAGE;

/*--------------------------------------------------------------------*/
SEND_MESSAGE: PROC;
/*--------------------------------------------------------------------*/

/*      -------------------------------------------------------------*/
/*      ensure we're in right state to send a message                 */
/*      -------------------------------------------------------------*/
        CALL TDSTATUS (GWL_PROC, GWL_RC,
                       GWL_STATUS_NR,
                       GWL_STATUS_DONE,
                       GWL_STATUS_COUNT,
```

```
                        GWL_STATUS_COMM,
                        GWL_STATUS_RETURN_CODE,
                        GWL_STATUS_SUBCODE);

        IF GWL_RC = TDS_OK THEN
        DO;
            IF GWL_STATUS_COMM = TDS_RECEIVE THEN

            CALL TDSNDMSG (GWL_PROC, GWL_RC,
                           MSG_TYPE, MSG_NR,
                           MSG_SEVERITY,
                           TDS_ZERO,
                           TDS_ZERO,
                           MSG_RPC, MSG_RPC_L,
                           MSG_TEXT, MSG_TEXT_L);
        END;

        RETURN;

END SEND_MESSAGE;

/*--------------------------------------------------------------------*/
LOCAL_TRACING: PROC;
/*--------------------------------------------------------------------*/
        CALL TDSETSPT (GWL_INIT_HANDLE, GWL_RC,
                       TRACING_SET_SW,
                       GWL_SETSPT_TRACE_LEVEL,
                       GWL_SETSPT_RPC_NAME,
                       GWL_SETSPT_RPC_NAME_L);

        RETURN;

END LOCAL_TRACING;

/*--------------------------------------------------------------------*/
CICS_ROLLBACK: PROC;
/*--------------------------------------------------------------------*/
        EXEC CICS SYNCPOINT
                  ROLLBACK
                  RESP(CICS_RESPONSE);

        RETURN;

END CICS_ROLLBACK;

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

```
END_PROGRAM:
/*-------------------------------------------------------------------*/
        IF (TRACING_SET_SW) THEN
        DO;
            TRACING_SET_SW = TRACING_RESET;
            CALL LOCAL_TRACING;
        END;

        IF SEND_DONE = SEND_DONE_OK THEN
            WRK_DONE_STATUS = TDS_DONE_COUNT;

        ELSE
        DO;
            WRK_DONE_STATUS = TDS_DONE_ERROR;
            SQLERRD(3)      = 0;
        END;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       WRK_DONE_STATUS,
                       SQLERRD(3),
                       TDS_ZERO,
                       TDS_ENDRPC);

        CALL TDFREE (GWL_PROC, GWL_RC);
        EXEC CICS RETURN;

END SYSAMP1;
```

APPENDIX D

Sample RPC Application for IMS TM (Explicit)

This appendix contains a sample mainframe server application program that runs in explicit mode under IMS TM and processes a client RPC. The PL/1 program listed here is included on the Open ServerConnect API tape.

The purpose of this sample program is to demonstrate the use of Gateway-Library functions in IMS TM programs, particularly those designed to handle remote procedure calls from a client. In some cases, one Gateway-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 ServerConnect package.

Sample program SYIXSAM1

```
SYIXAM1: PROC OPTIONS(MAIN, NOEXECOPS);
 /******* SYIXSAM1 - RPC REQUEST APPLICATION - PL/1 - IMS ************/
 /*                                                                   */
 /*  TRANID:        SYIXSAM1                                          */
 /*  PROGRAM:       SYIXSAM1                                          */
 /*  PLAN NAME:     n/a                                               */
 /*  FILES:         n/a                                               */
 /*  TABLES:        n/a                                               */
 /*                                                                   */
 /*  The purpose of the program is to stress test the IMS Open Server*/
 /*  This program is can be invoked by isql. The first parameter is a*/
 /*  one byte character parm that is used to set up a dummy results*/
 /*  row.  The second parameter is the number of rows to return. To  */
 /*  execute, setup an RPC, SYIXSAM1 to invoke mainframe transaction */
 /*  SYIXSAM1. Remote_lu is the VTAM APPLID of the LU6.2              */
 /*  APPLID that IMS is known by if running implicit mode. Security  */
```

```
/*  is none, user id, or both.                                          */
/*                                                                      */
/* >isql -Usa -P -Sservername                                           */
/*                                                                      */
/* >exec sgw_addrpc SYIXSAM1,SYIXSAM1,remote_lu,security                */
/*                                                                      */
/* >go                                                                  */
/*                                                                      */
/* >exec SYIXSAM1 X, 100                                                */
/*                                                                      */
/* >go                                                                  */
/*                                                                      */
/*  This tran returns a 80 byte row containing the name of              */
/*  the client that initiated the RPC and a 71 byte pattern.            */
/*                                                                      */
/*  Server Library calls:                                               */
/*    TDACCEPT      accept request from client                          */
/*    TDESCRIB      describe a column                                   */
/*    TDGETREQ      Get next RPC                                        */
/*    TDFREE        free TDPROC structure                               */
/*    TDINIT        establish environment                               */
/*    TDRCVPRM      retrieve rpc parameter from client                  */
/*    TDRESULT      describe next communication                         */
/*    TDSETPT       set program type                                    */
/*    TDSNDDON      send results-completion to client                   */
/*    TDSNDMSG      send message to client                              */
/*    TDSNDROW      send row to client                                  */
/*    TDTERM        free all storage                                    */
/*                                                                      */
/************************************************************************/
 DCL     P_PCBTERM              POINTER;
 DCL     P_ALTPCB               POINTER;

 DCL     PLIXOPT CHAR(50) VAR INIT('NOSPIE,NOSTAE')
         STATIC EXTERNAL;

/*----------------------------------------------------------------------*/
/*        IMS TERMINAL PCB                                              */
/*----------------------------------------------------------------------*/

 DCL  1  PCBTERM BASED(P_PCBTERM),
         5  TERMNAME             CHAR(8),
         5  RESERVED             BIT(16),
         5  STATUS               CHAR(2),
         5  DATE                 FIXED(7),
         5  TIME                 FIXED(7),
```

```
         5  MSGCOUNT             FIXED BIN(31),
         5  MODNAME              CHAR(8);
/*****************************************************************/
        %INCLUDE SYGWPLI;

/*---------------------------------------------------------------*/
/*        SERVER LIB ROUTINES DECLARATIONS                       */
/*---------------------------------------------------------------*/
        DCL
            SRRCMIT  ENTRY OPTIONS(INTER ASSEMBLER),
            TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
            TDESCRIB ENTRY OPTIONS(INTER ASSEMBLER),
            TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
            TDGETREQ ENTRY OPTIONS(INTER ASSEMBLER),
            TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
            TDRCVPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDRESULT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETPT  ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDROW ENTRY OPTIONS(INTER ASSEMBLER),
            TDTERM   ENTRY OPTIONS(INTER ASSEMBLER);

/*---------------------------------------------------------------*/
/*      AIB INTERFACE DECLARATIONS                               */
/*---------------------------------------------------------------*/
        DCL AIBTDLI ENTRY OPTIONS(INTER ASSEMBLER);

        DCL TWO                          FIXED BIN(31) INIT(2);
        DCL APSB                         CHAR(4) INIT('APSB');
        DCL DPSB                         CHAR(4) INIT('DPSB');

        DCL
            01  AIB,
                5  AIBID                 CHAR(08),
                5  AIBLEN                FIXED BIN(31),
                5  AIBSFUNC              CHAR(08),
                5  AIBRSNM1              CHAR(08),
                5  FILLER1               CHAR(16),
                5  AIBOALEN              FIXED BIN(31),
                5  AIBOAUSE              FIXED BIN(31),
                5  FILLER2               CHAR(12),
                5  AIBRETRN              FIXED BIN(31),
                5  AIBREASN              FIXED BIN(31),
                5  FILLER3               CHAR(04),
                5  AIBRSA1               POINTER,
```

```
            5  FILLER4                  CHAR(44);

        DCL
            01  PCB_ADDRESS BASED(AIBRSA1),
                5  PCB1_PTR              POINTER,
                5  PCB2_PTR              POINTER,
                5  PCB3_PTR              POINTER;

/*---------------------------------------------------------------------*/
/*        BUILT IN FUNCTIONS DECLARATIONS                              */
/*---------------------------------------------------------------------*/
        DCL
            ADDR    BUILTIN,
            NULL    BUILTIN,
            STG     BUILTIN;

/*---------------------------------------------------------------------*/
/*        WORK AREAS                                                   */
/*---------------------------------------------------------------------*/
        DCL
            01  GW_LIB_MISC_FIELDS,
                05  GWL_SPA_PTR          PTR,
                05  GWL_PROC             PTR,
                05  GWL_INIT_HANDLE      PTR,
                05  GWL_RC               FIXED BIN(31),
                05  GWL_SEND_DONE        FIXED BIN(31),
                05  GWL_WAIT_OPTION      FIXED BIN(31),
                05  GWL_REQ_TYPE         FIXED BIN(31),
                05  GWL_PROG_TYPE        CHAR(04) INIT('MPP '),
                05  GWL_TRAN_NAME        CHAR(30);

        DCL     CPIC_RC                  FIXED BIN(31) INIT(0);

        DCL
            01  SNA_FIELDS,
                05  SNA_SUBC             FIXED BIN(31),
                05  SNA_CONNECTION_NAME  CHAR(08)      INIT(' ');

        DCL
            01  PARM_FIELDS,
                05  PARM_PATTERN         CHAR(01),
                05  BANANA               CHAR(06) INIT('BANANA'),
                05  I                    FIXED BIN(31) INIT(1),
                05  PARM_NR_ROWS         FIXED BIN(31) INIT(1),
                05  PARM_ID1             FIXED BIN(31) INIT(1),
                05  PARM_ID2             FIXED BIN(31) INIT(2),
```

```
            05  PARM_L                 FIXED BIN(31);

        DCL     WROW CHAR(80);

        DCL
            01  WROW_R DEFINED WROW,
                05  WROW_LU            CHAR(08),
                05  WROW_FILL          CHAR(01),
                05  WROW_PATTERN (71)  CHAR(01);
        DCL
            01  COUNTER_FIELDS,
                05  CTR_COLUMN         FIXED BIN(31) INIT(1),
                05  CTR_ROWS           FIXED BIN(31) INIT(0);

        DCL
            01  DESCRIBE_BIND_FIELDS,
                05  DB_NULL_INDICATOR  FIXED BIN(15) INIT(0);

        DCL
           01   WORK_FIELDS,
                05  WRKLEN1            FIXED BIN(31),
                05  WRKLEN2            FIXED BIN(31);

        DCL  SYSPRINT FILE EXTERNAL;

        DCL  01  ERROR_MSG,
                 05  FILL_1            CHAR(01),
                 05  CALL_NAME         CHAR(09),
                 05  FILL_2            CHAR(01),
                 05  CALL_MSG          CHAR(20),
                 05  CALL_RC           PIC '-ZZZ9',
                 05  FILL_3            CHAR(01),
                 05  CALL_RC2          PIC '-ZZZ9',
                 05  FILL_4            CHAR(01);

        DCL
             01  SWITCHES,
                 05  FALSE             BIT(01) INIT('0'B),
                 05  TRUE              BIT(01) INIT('1'B),
                 05  ALL_DONE          BIT(01) INIT('0'B),
                 05  ALL_DONE_YES      BIT(01) INIT('1'B),
                 05  MORE_MSGS         BIT(01) INIT('1'B);

/*-------------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-------------------------------------------------------------------*/
```

```
         P_PCBTERM = NULL;
         GWL_PROC = NULL;
         GWL_SPA_PTR = NULL;
         GWL_INIT_HANDLE = NULL;
         GWL_RC = 0;
         MORE_MSGS = TRUE;

/*-------------------------------------------------------------------*/
/*      ALLOCATE AIB for IMS LOG access                              */
/*-------------------------------------------------------------------*/
         AIBID = 'DFSAIB  ';
         AIBRSNM1 = 'SYIPSAM1';
         AIBLEN = STG(AIB);

         CALL AIBTDLI (TWO, APSB, AIB);

         IF AIBRETRN = 0 THEN
               P_PCBTERM = PCB1_PTR;
         ELSE
            DO;

               CALL_NAME = 'APSB';
               CALL_MSG = 'APSB CALL ERROR RC=';
               CALL_RC  = AIBRETRN;
               CALL_RC2 = AIBREASN;
               PUT FILE(SYSPRINT) DATA(ERROR_MSG);
               RETURN;
            END;

/*      ------------------------------------------------------------*/
/*      establish gateway environment                              */
/*      ------------------------------------------------------------*/
         CALL TDINIT (P_PCBTERM, GWL_RC, GWL_INIT_HANDLE);

         IF GWL_RC ^= 0 THEN
            DO;
               CALL_NAME = 'TDINIT';
               CALL DISP_ERROR;
               RETURN;
            END;

/*      ------------------------------------------------------------*/
/*      set program type to EXPL                                   */
/*      ------------------------------------------------------------*/
         GWL_PROG_TYPE = 'EXPL';
```

```
        CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                      GWL_SPA_PTR, TDS_NULL, TDS_NULL);
        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDSETPT';
              CALL DISP_ERROR;
              RETURN;
           END;

/*      ------------------------------------------------------------*/
/*      accept client request                                       */
/*      ------------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);

        IF GWL_RC ^= 0 THEN
           DO;
              MORE_MSGS = FALSE;
              CALL_NAME = 'TDACCEPT';
              CALL DISP_ERROR;
              RETURN;
           END;

   DO WHILE(MORE_MSGS);

/*------------------------------------------------------------------*/
/*      GET PATTERN TO SEND                                         */
/*------------------------------------------------------------------*/
        WRKLEN1 = STG(PARM_PATTERN);

        CALL TDRCVPRM (GWL_PROC, GWL_RC, PARM_ID1,
                       PARM_PATTERN, TDSCHAR, WRKLEN1, PARM_L);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDRCVPRM';
              CALL DISP_ERROR;
           END;

/*------------------------------------------------------------------*/
/*      GET NUMBER OF ROWS TO SEND                                  */
/*------------------------------------------------------------------*/
        WRKLEN1 = STG(PARM_NR_ROWS);
```

```
         CALL TDRCVPRM (GWL_PROC, GWL_RC,
                        PARM_ID2,
                        PARM_NR_ROWS, TDSINT4, WRKLEN1, PARM_L);

         IF GWL_RC ^= 0 THEN
            DO;
               CALL_NAME = 'TDRCVPRM2';
               CALL DISP_ERROR;
            END;

         IF PARM_NR_ROWS = 0 THEN
            GO TO SEND_DONE;

SETUP_REPLY_COLUMN:

         WRKLEN1                = 80;
         WRKLEN2                = STG(BANANA);

         CALL TDESCRIB (GWL_PROC, GWL_RC,
                        CTR_COLUMN,
                        TDSCHAR, WRKLEN1, WROW,
                        DB_NULL_INDICATOR, TDS_FALSE,
                        TDSCHAR, WRKLEN1, BANANA, WRKLEN2);

         IF GWL_RC ^= 0 THEN
            DO;
               CALL_NAME = 'TDESCRIB';
               CALL DISP_ERROR;
            END;

         WROW = ' ';

         WROW_LU = 'BANANA  ';

         DO I = 1 TO 71;

            WROW_PATTERN (I) = PARM_PATTERN;

         END;

/*---------------------------------------------------------------------*/
SEND_REPLY_ROWS:
/*---------------------------------------------------------------------*/
         ALL_DONE = '0'B;
         CTR_ROWS = 0;
```

```
        DO WHILE(^ ALL_DONE);

            CALL TDSNDROW (GWL_PROC, GWL_RC);

            IF GWL_RC = TDS_CANCEL_RECEIVED THEN
                 ALL_DONE = ALL_DONE_YES;
            ELSE
            IF GWL_RC ^= 0 THEN
               DO;
                  CALL_NAME = 'TDSNDROW';
                  CALL DISP_ERROR;
               END;
            ELSE
               CTR_ROWS = CTR_ROWS + 1;

            IF CTR_ROWS >= PARM_NR_ROWS THEN
               ALL_DONE = ALL_DONE_YES;

        END;

/*-------------------------------------------------------------------*/
SEND_DONE:
/*-------------------------------------------------------------------*/

        IF GWL_RC = TDS_CANCEL_RECEIVED THEN
           DO;
             MORE_MSGS = FALSE;
             GWL_SEND_DONE = TDS_ENDRPC;
           END;
        ELSE
        IF PARM_NR_ROWS = 0 THEN
           DO;
             MORE_MSGS = FALSE;
             GWL_SEND_DONE = TDS_ENDRPC;
           END;
        ELSE
           GWL_SEND_DONE = TDS_ENDREPLY;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       TDS_DONE_COUNT,
                       CTR_ROWS, TDS_ZERO,
                       GWL_SEND_DONE);

            IF GWL_RC ^= 0 THEN
               DO;
                  CALL_NAME = 'TDSNDDON';
```

```
                CALL DISP_ERROR;
              END;

       IF PARM_NR_ROWS > 0 THEN
          DO;
           GWL_WAIT_OPTION = TDS_TRUE;
           GWL_REQ_TYPE = 0;
           GWL_TRAN_NAME = ' ';

           CALL TDGETREQ (GWL_PROC, GWL_RC,
                          GWL_WAIT_OPTION,
                          GWL_REQ_TYPE,
                          GWL_TRAN_NAME);

           SELECT (GWL_RC);
             WHEN(TDS_OK);
             WHEN(TDS_CONNECTION_TERMINATED)
                 DO;
                    MORE_MSGS = FALSE;
                    CALL FREE_STORAGE;
                 END;
             WHEN(TDS_RESULTS_COMPLETE)
                 DO;
                    MORE_MSGS = FALSE;
                    CALL FREE_STORAGE;
                 END;
             OTHERWISE
                 DO;
                    MORE_MSGS = FALSE;
                    CALL_NAME = 'TDGETREQ';
                    CALL DISP_ERROR;
                 END;
            END;    /* SELECT */

   END;  /* IF */

END;    /* DO WHILE MORE_MSGS = TRUE */

CALL SRRCMIT(CPIC_RC);

IF CPIC_RC ^= 0 then
   DO;

      CALL_NAME = 'SRRCMIT';
      CALL_MSG = 'SRRCMIT ERROR RC=';
      CALL_RC  = CPIC_RC;
```

```
         PUT FILE(SYSPRINT) DATA(ERROR_MSG);

      END;

   AIBID = 'DFSAIB  ';
   AIBRSNM1 = 'SYIPSAM1';
   AIBLEN = STG(AIB);
   CALL AIBTDLI (TWO, DPSB, AIB);

   IF AIBRETRN ^= 0 THEN
      DO;
         CALL_NAME = 'DPSB';
         CALL_MSG = 'DPSB CALL ERROR RC=';
         CALL_RC  = AIBRETRN;
         CALL_RC2 = AIBREASN;
         PUT FILE(SYSPRINT) DATA(ERROR_MSG);
      END;

   IF PARM_NR_ROWS = 0 then
      CALL FREE_STORAGE;
DISP_ERROR: PROC;
        CALL_MSG = 'ERROR IN CALL RC=';
        CALL_RC  = GWL_RC;
        PUT FILE(SYSPRINT) DATA(ERROR_MSG);
        CALL FREE_STORAGE;

END DISP_ERROR;
FREE_STORAGE: PROC;

        CALL TDFREE (GWL_PROC, GWL_RC);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDFREE  ';
              CALL_MSG = 'ERROR IN CALL RC=';
              CALL_RC  = GWL_RC;
              PUT FILE(SYSPRINT) DATA(ERROR_MSG);
           END;

        CALL TDTERM (GWL_INIT_HANDLE, GWL_RC);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDTERM  ';
              CALL_MSG = 'ERROR IN CALL RC=';
              CALL_RC  = GWL_RC;
```

```
               PUT FILE(SYSPRINT) DATA(ERROR_MSG);
            END;

  END FREE_STORAGE;

  END SYIXAM1;
```

APPENDIX E

Sample RPC Application for IMS TM (Implicit)

This appendix contains a sample mainframe server application program that runs in implicit mode under IMS TM and processes a series of client RPCs from the Open Client program SYM1. The PL/1 program listed here is included on the Open ServerConnect API tape.

The purpose of this sample program is to demonstrate the use of Gateway-Library functions in IMS TM programs, particularly those designed to handle remote procedure calls from a client. In some cases, one Gateway-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 ServerConnect package.

Sample program SYIPSAM1

```
SYIPAM1: PROC(P_PCBTERM) OPTIONS(MAIN, NOEXECOPS);
 /****** SYIPSAM1 - RPC REQUEST APPLICATION - PL/1 - IMS ************/
 /*                                                                  */
 /*  TRANID:        SYM1                                             */
 /*  PROGRAM:       SYIPSAM1                                         */
 /*  PLAN NAME:     n/a                                              */
 /*  FILES:         n/a                                              */
 /*  TABLES:        n/a                                              */
 /*                                                                  */
 /* The purpose of the program is to stress test the IMS Open Server.*/
 /*  This program is executed via isql.  The first parameter is a    */
 /*  a one byte character parm that is used to set up a dummy results*/
 /*  row.  The second parameter is the number of rows to return. To  */
 /*  execute enter the following commands:                           */
 /*                                                                  */
 /* >isql -Usa -P -Sservername                                       */
```

```
/*                                                                    */
/* >exec SYM1 X, 100                                                  */
/*                                                                    */
/*  This tran returns a 80 byte row containing the LTERM name of      */
/*  the client that initiated the RPC and a 71 byte pattern.          */
/*                                                                    */
/*  Server Library calls:                                             */
/*    TDACCEPT      accept request from client                        */
/*    TDESCRIB      describe a column                                 */
/*    TDGETREQ      Get next RPC                                      */
/*    TDFREE        free TDPROC structure                             */
/*    TDINIT        establish environment                             */
/*    TDRCVPRM      retrieve rpc parameter from client                */
/*    TDRESULT      describe next communication                       */
/*    TDSETPT       set program type                                  */
/*    TDSNDDON      send results-completion to client                 */
/*    TDSNDMSG      send message to client                            */
/*    TDSNDROW      send row to client                                */
/*    TDTERM        free all storage                                  */
/*                                                                    */
/********************************************************************/

 DCL     PLIXOPT CHAR(50) VAR INIT('NOSPIE,NOSTAE')
         STATIC EXTERNAL;

 DCL     P_PCBTERM               POINTER;
 DCL     P_ALTPCB                POINTER;

/*------------------------------------------------------------------*/
/*       POINTER TO ACTUAL PCB ADDRESS                              */
/*------------------------------------------------------------------*/

 DCL   P_PCBADDR POINTER BASED(P_PCBTERM);

/*------------------------------------------------------------------*/
/*         IMS TERMINAL PCB                                         */
/*------------------------------------------------------------------*/

 DCL  1  PCBTERM BASED(P_PCBTERM),
         5  TERMNAME         CHAR(8),
         5  RESERVED         BIT(16),
         5  STATUS           CHAR(2),
         5  DATE             FIXED(7),
         5  TIME             FIXED(7),
         5  MSGCOUNT         FIXED BIN(31),
         5  MODNAME          CHAR(8);
```

```
/*****************************************************************/
/*        SERVER LIBRARY PL/1 COPY BOOK                          */
/*****************************************************************/
        %INCLUDE SYGWPLI;

/*---------------------------------------------------------------*/
/*        SERVER LIB ROUTINES DECLARATIONS                       */
/*---------------------------------------------------------------*/
        DCL
            TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
            TDESCRIB ENTRY OPTIONS(INTER ASSEMBLER),
            TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
            TDGETREQ ENTRY OPTIONS(INTER ASSEMBLER),
            TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
            TDRCVPRM ENTRY OPTIONS(INTER ASSEMBLER),
            TDRESULT ENTRY OPTIONS(INTER ASSEMBLER),
            TDSETPT  ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
            TDSNDROW ENTRY OPTIONS(INTER ASSEMBLER),
            TDTERM   ENTRY OPTIONS(INTER ASSEMBLER);

/*---------------------------------------------------------------*/
/*        BUILT IN FUNCTIONS DECLARATIONS                        */
/*---------------------------------------------------------------*/
        DCL
            ADDR    BUILTIN,
            NULL    BUILTIN,
            STG     BUILTIN;

/*---------------------------------------------------------------*/
/*        WORK AREAS                                             */
/*---------------------------------------------------------------*/
        DCL
            01  GW_LIB_MISC_FIELDS,
                05  GWL_SPA_PTR          PTR,
                05  GWL_PROC             PTR,
                05  GWL_INIT_HANDLE      PTR,
                05  GWL_RC               FIXED BIN(31),
                05  GWL_WAIT_OPTION      FIXED BIN(31),
                05  GWL_REQ_TYPE         FIXED BIN(31),
                05  GWL_PROG_TYPE        CHAR(04) INIT('MPP '),
                05  GWL_TRAN_NAME        CHAR(30);

        DCL
```

```
    01  SNA_FIELDS,
        05  SNA_SUBC              FIXED BIN(31),
        05  SNA_CONNECTION_NAME   CHAR(08)      INIT(' ');

DCL
    01  PARM_FIELDS,
        05  PARM_PATTERN          CHAR(01),
        05  BANANA                CHAR(06) INIT('BANANA'),
        05  I                     FIXED BIN(31) INIT(1),
        05  PARM_NR_ROWS          FIXED BIN(31) INIT(1),
        05  PARM_ID1              FIXED BIN(31) INIT(1),
        05  PARM_ID2              FIXED BIN(31) INIT(2),
        05  PARM_L                FIXED BIN(31);

DCL     WROW CHAR(80);
DCL
    01  WROW_R DEFINED WROW,
        05  WROW_LU               CHAR(08),
        05  WROW_FILL             CHAR(01),
        05  WROW_PATTERN (71)     CHAR(01);
DCL
    01  COUNTER_FIELDS,
        05  CTR_COLUMN            FIXED BIN(31) INIT(1),
        05  CTR_ROWS              FIXED BIN(31) INIT(0);

DCL
    01  DESCRIBE_BIND_FIELDS,
        05  DB_NULL_INDICATOR     FIXED BIN(15) INIT(0);

DCL
   01   WORK_FIELDS,
        05  WRKLEN1               FIXED BIN(31),
        05  WRKLEN2               FIXED BIN(31);

DCL  SYSPRINT FILE EXTERNAL;

DCL  01  ERROR_MSG,
         05  FILL_1               CHAR(01),
         05  CALL_NAME            CHAR(09),
         05  FILL_2               CHAR(01),
         05  CALL_MSG             CHAR(20),
         05  CALL_RC              PIC '-ZZZ9',
         05  FILL_3               CHAR(01);

DCL
     01  SWITCHES,
```

```
                05  FALSE                BIT(01) INIT('0'B),
                05  TRUE                 BIT(01) INIT('1'B),
                05  ALL_DONE             BIT(01) INIT('0'B),
                05  ALL_DONE_YES         BIT(01) INIT('1'B),
                05  MORE_MSGS            BIT(01) INIT('1'B);

/*-------------------------------------------------------------------*/
INITIALIZE_PROGRAM:
/*-------------------------------------------------------------------*/

        GWL_PROC = NULL;
        GWL_SPA_PTR = NULL;
        GWL_INIT_HANDLE = NULL;
        GWL_RC = 0;
        MORE_MSGS = TRUE;

/*      ---------------------------------------------------------*/
/*      establish gateway environment                            */
/*      ---------------------------------------------------------*/
        CALL TDINIT (P_PCBADDR, GWL_RC, GWL_INIT_HANDLE);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDINIT';
              CALL DISP_ERROR;
           END;

/*      ---------------------------------------------------------*/
/*      set program type to MPP                                  */
/*      ---------------------------------------------------------*/
        CALL TDSETPT (GWL_INIT_HANDLE, GWL_RC, GWL_PROG_TYPE,
                      GWL_SPA_PTR, TDS_NULL, TDS_NULL);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDSETPT';
              CALL DISP_ERROR;
           END;

/*      ---------------------------------------------------------*/
/*      accept client request                                    */
/*      ---------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);
```

```
IF GWL_RC ^= 0 THEN
             DO;
                MORE_MSGS = FALSE;
                CALL_NAME = 'TDACCEPT';
                CALL DISP_ERROR;
             END;

  DO WHILE(MORE_MSGS);

  /*-------------------------------------------------------------------*/
  /*      GET PATTERN TO SEND                                          */
  /*-------------------------------------------------------------------*/
          WRKLEN1 = STG(PARM_PATTERN);

          CALL TDRCVPRM (GWL_PROC, GWL_RC,
                         PARM_ID1,
                         PARM_PATTERN, TDSCHAR, WRKLEN1, PARM_L);

          IF GWL_RC ^= 0 THEN
             DO;
                CALL_NAME = 'TDRCVPRM';
                CALL DISP_ERROR;
             END;

  /*-------------------------------------------------------------------*/
  /*      GET NUMBER OF ROWS TO SEND                                   */
  /*-------------------------------------------------------------------*/
          WRKLEN1 = STG(PARM_NR_ROWS);

          CALL TDRCVPRM (GWL_PROC, GWL_RC,
                         PARM_ID2,
                         PARM_NR_ROWS, TDSINT4, WRKLEN1, PARM_L);

          IF GWL_RC ^= 0 THEN
             DO;
                CALL_NAME = 'TDRCVPRM2';
                CALL DISP_ERROR;
             END;

          IF PARM_NR_ROWS = 0 THEN
             GO TO SEND_DONE;

  SETUP_REPLY_COLUMN:

          WRKLEN1                = 80;
          WRKLEN2                = STG(BANANA);
```

```
        CALL TDESCRIB (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       TDSCHAR, WRKLEN1, WROW,
                       DB_NULL_INDICATOR, TDS_FALSE,
                       TDSCHAR, WRKLEN1, BANANA, WRKLEN2);

        IF GWL_RC ^= 0 THEN
           DO;
              CALL_NAME = 'TDESCRIB';
              CALL DISP_ERROR;
           END;

        WROW = ' ';

        WROW_LU = PCBTERM.TERMNAME;

        DO I = 1 TO 71;

           WROW_PATTERN (I) = PARM_PATTERN;

        END;

/*-------------------------------------------------------------------*/
SEND_REPLY_ROWS:
/*-------------------------------------------------------------------*/
        ALL_DONE = '0'B;
        CTR_ROWS = 0;

        IF PARM_NR_ROWS = 0 THEN
           ALL_DONE = ALL_DONE_YES;
        ELSE
            DO WHILE(^ ALL_DONE);

                CALL TDSNDROW (GWL_PROC, GWL_RC);

                IF GWL_RC ^= 0 THEN
                   DO;
                      CALL_NAME = 'TDSNDROW';
                      CALL DISP_ERROR;
                   END;
                ELSE
                   CTR_ROWS = CTR_ROWS + 1;

                IF CTR_ROWS >= PARM_NR_ROWS THEN
                   ALL_DONE = ALL_DONE_YES;
```

```
            END; /* END DO WHILE */

/*--------------------------------------------------------------------*/
SEND_DONE:
/*--------------------------------------------------------------------*/

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       TDS_DONE_COUNT,
                       CTR_ROWS,
                       TDS_ZERO,
                       TDS_ENDRPC);

            IF GWL_RC ^= 0 THEN
               DO;
                  CALL_NAME = 'TDSNDDON';
                  CALL DISP_ERROR;
               END;

        IF PARM_NR_ROWS = 0 THEN
            MORE_MSGS = FALSE;
        ELSE
           DO;
              GWL_WAIT_OPTION = TDS_TRUE;
              GWL_REQ_TYPE = 0;
              GWL_TRAN_NAME = ' ';

              CALL TDGETREQ (GWL_PROC, GWL_RC,
                             GWL_WAIT_OPTION,
                             GWL_REQ_TYPE,
                             GWL_TRAN_NAME);

              SELECT (GWL_RC);
                WHEN(TDS_OK);
                WHEN(TDS_CONNECTION_TERMINATED)
                       MORE_MSGS = FALSE;
                WHEN(TDS_RESULTS_COMPLETE)
                       MORE_MSGS = FALSE;
                OTHERWISE
                    DO;
                       MORE_MSGS = FALSE;
                       CALL_NAME = 'TDGETREQ';
                       CALL DISP_ERROR;
                    END;
               END;  /* SELECT */
           END;    /* END ELSE */
```

```
END;  /* DO WHILE MORE_MSGS = TRUE */

CALL FREE_STORAGE;

DISP_ERROR: PROC;

   MORE_MSGS = FALSE;
   CALL_MSG = 'ERROR IN CALL RC=';
   CALL_RC  = GWL_RC;
   PUT FILE(SYSPRINT) DATA(ERROR_MSG);

END DISP_ERROR;

FREE_STORAGE: PROC;

  RETURN;
  CALL TDFREE (GWL_PROC, GWL_RC);

  IF GWL_RC ^= 0 THEN
     DO;
        CALL_NAME = 'TDFREE  ';
        CALL_MSG = 'ERROR IN CALL RC=';
        CALL_RC  = GWL_RC;
        PUT FILE(SYSPRINT) DATA(ERROR_MSG);
     END;

  CALL TDTERM (GWL_INIT_HANDLE, GWL_RC);

  IF GWL_RC ^= 0 THEN
     DO;
        CALL_NAME = 'TDTERM  ';
        CALL_MSG = 'ERROR IN CALL RC=';
        CALL_RC  = GWL_RC;
        PUT FILE(SYSPRINT) DATA(ERROR_MSG);
     END;

END FREE_STORAGE;

END SYIPAM1;
```

APPENDIX F

Sample Mixed-Mode Application

This appendix contains a sample PL/1 application program that uses both Client-Library and Gateway-Library functions. In other words, this program acts as both client and server.

This sample program is provided on the Open ServerConnect tape. Running SYCTSAX4 requires Open ServerConnect. It uses PL/1 and Gateway-Library.

Sample program SYCTSAX4

```
SYCTSAX4: PROC OPTIONS(MAIN REENTRANT);
 /*       @(#) syctsax4.pli 1.1 4/18/96       */

 /****** SYCTSAX4 - Open Server, Open Client - PL/1 - CICS ***********/
 /*                                                                  */
 /*  TRANID:        SYX4                                             */
 /*  PROGRAM:       SYCTSAX4                                         */
 /*  TABLE:         SYBASE.SAMPLETB                                  */
 /*                                                                  */
 /*  The purpose of the program is primarily to demonstrate the      */
 /*  ability to act as a server and a client within one program.     */
 /*                                                                  */
 /*  This program is invoked via an RPC request and will in turn     */
 /*  execute a language request against a server and return the      */
 /*  results back to the client.                                     */
 /*                                                                  */
 /*  It will issue the following SQL statement:                      */
 /*                                                                  */
 /*   "SELECT FIRSTNME FROM SYBASE.SAMPLETB"                         */
 /*                                                                  */
 /*  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.                                    */
```

```
/*                                                                  */
/*  Server Library calls:                                           */
/*                                                                  */
/*    TDACCEPT      accept request from client                      */
/*    TDESCRIB      describe a column in the result row             */
/*    TDFREE        free TDPROC structure                           */
/*    TDINFPRM      get information about one rpc parameter         */
/*    TDINIT        establish environment                           */
/*    TDNUMPRM      get total nr of rpc parameters                  */
/*    TDRCVPRM      retrieve rpc parameter from client              */
/*    TDSNDDON      send results-completion to client               */
/*    TDSNDMSG      send error messages back to the client          */
/*    TDSNDROW      send a row of data back to the client           */
/*                                                                  */
/*  Open Client calls:                                              */
/*                                                                  */
/*    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                        */
/*    CTBDIAG       retrieve SQLCODE messages                       */
/*    CTBEXIT       exit client library                             */
/*    CTBFETCH      fetch result data                               */
/*    CTBINIT       init client library                             */
/*    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          */
/*                                                                  */
/*                                                                  */
/********************************************************************/

/*------------------------------------------------------------------*/
/* CLIENT LIBRARY PL/1 COPY BOOK                                    */
/*------------------------------------------------------------------*/
```

```
%INCLUDE CTPUBLIC;
/*--------------------------------------------------------------------*/
  /* SERVER LIBRARY PL/1 COPY BOOK                                     */
  /*--------------------------------------------------------------------*/
%INCLUDE SYGWPLI;
/*--------------------------------------------------------------------*/
  /* CICS Standard Attention Identifiers PL/1 Copy Book                */
  /*--------------------------------------------------------------------*/
%INCLUDE DFHAID;
/*--------------------------------------------------------------------*/
  /* SERVER LIBRARY ROUTINE DECLARATIONS                               */
  /*--------------------------------------------------------------------*/
         DCL
             TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
             TDESCRIB ENTRY OPTIONS(INTER ASSEMBLER),
             TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
             TDINFPRM ENTRY OPTIONS(INTER ASSEMBLER),
             TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
             TDNUMPRM ENTRY OPTIONS(INTER ASSEMBLER),
             TDRCVPRM ENTRY OPTIONS(INTER ASSEMBLER),
             TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
             TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
             TDSNDROW ENTRY OPTIONS(INTER ASSEMBLER);
/*********************************************************************/
/* CLIENT LIBRARY ROUTINE DECLARATIONS                               */
/*********************************************************************/
         DCL
             CSBCTXDR     ENTRY OPTIONS(INTER ASSEMBLER),
             CTBBIND      ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCLOSE     ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCMDALLOC  ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCMDDROP   ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCOMMAND   ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCONALLOC  ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCONDROP   ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCONPROPS  ENTRY OPTIONS(INTER ASSEMBLER),
             CTBCONNECT   ENTRY OPTIONS(INTER ASSEMBLER),
             CTBDIAG      ENTRY OPTIONS(INTER ASSEMBLER),
             CTBEXIT      ENTRY OPTIONS(INTER ASSEMBLER),
             CTBFETCH     ENTRY OPTIONS(INTER ASSEMBLER),
             CTBINIT      ENTRY OPTIONS(INTER ASSEMBLER),
             CTBRESULTS   ENTRY OPTIONS(INTER ASSEMBLER),
             CTBSEND      ENTRY OPTIONS(INTER ASSEMBLER);
/*********************************************************************/
/* BUILT-IN FUNCTION DECLARATIONS                                    */
/*********************************************************************/
```

```
          DCL
              ADDR      BUILTIN,
              LENGTH    BUILTIN,
              STG       BUILTIN,
              SUBSTR    BUILTIN;
/*****************************************************************/
/* WORK AREAS SERVER LIBRARY                                     */
/*****************************************************************/
          DCL
              01  GW_LIB_MISC_FIELDS,
                  05  GWL_TDPROC              PTR,
                  05  GWL_RC                  FIXED BIN(31);
DCL
              01  PARM_FIELDS,
                  05  PF_PARM_ID              FIXED BIN(31),
                  05  PF_DATATYPE             FIXED BIN(31),
                  05  PF_ACTUAL_DATA_LENGTH   FIXED BIN(31),
                  05  PF_MAX_DATA_LENGTH      FIXED BIN(31),
                  05  PF_PARM_STATUS          FIXED BIN(31),
                  05  PF_PARM_NAME            CHAR(30),
                  05  PF_PARM_NAME_LENGTH     FIXED BIN(31),
                  05  PF_USER_DATATYPE        FIXED BIN(31),
                  05  PF_NUM_OF_PARMS         FIXED BIN(31);
DCL
              01  SNA_FIELDS,
                  05  SNA_SUBC                FIXED BIN(31),
                  05  SNA_CONNECTION_NAME     CHAR(08) INIT(' ');
DCL
              01  WORK_FIELDS,
                  05  WRK_DONE_STATUS         FIXED BIN(31);
DCL
              01  DESCRIBE_FIELDS,
                  05  DF_COLUMN_NUMBER        FIXED BIN(31),
                  05  DF_HOST_VARIABLE_TYPE   FIXED BIN(31),
                  05  DF_HOST_VARIABLE_MAXLEN FIXED BIN(31),
                  05  DF_HOST_VARIABLE_NAME   PTR,
                  05  DF_NULL_INDICATOR_VAR   FIXED BIN(15),
                  05  DF_NULLS_ALLOWED        FIXED BIN(31),
                  05  DF_COLUMN_TYPE          FIXED BIN(31),
                  05  DF_COLUMN_MAXLEN        FIXED BIN(31),
                  05  DF_COLUMN_NAME          CHAR(30),
                  05  DF_COLUMN_NAME_LEN      FIXED BIN(31);
          DCL
              01  SNDMSG_FIELDS,
                  05  SF_MESSAGE_TYPE         FIXED BIN(31),
                  05  SF_MESSAGE_NUMBER       FIXED BIN(31),
```

```
                05  SF_SEVERITY             FIXED BIN(31),
                05  SF_ERROR_STATE          FIXED BIN(31),
                05  SF_LINE_ID              FIXED BIN(31),
                05  SF_TRANSACTION_ID       CHAR(4)     INIT('SYX4'),
                05  SF_TRANSACTION_ID_LEN   FIXED BIN(31),
                05  SF_MESSAGE_TEXT         CHAR(80),
                05  SF_MESSAGE_LENGTH       FIXED BIN(31);
/*-------------------------------------------------------------------*//
/* WORK AREAS OPEN CLIENT                                            */
/*-------------------------------------------------------------------*/
          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_RC                FIXED BIN(31);
          DCL
              01  PROPS_FIELDS,
                  05  PF_SERVER             CHAR(30),
                  05  PF_SERVER_LEN         FIXED BIN(31),
                  05  PF_USER               CHAR(08),
                  05  PF_USER_LEN           FIXED BIN(31),
                  05  PF_PWD                CHAR(30),
                  05  PF_PWD_LEN            FIXED BIN(31),
                  05  PF_OUTLEN             FIXED BIN(31),
                  05  PF_STRLEN             FIXED BIN(31),
                  05  PF_MSGLIMIT           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  FETCH_FIELDS,
                  05  FF_ROWS_READ          FIXED BIN(31),
                  05  FF_ROW_NUM            FIXED BIN(31)  INIT(0);
          DCL
              01  COLUMN_FIELDS,
                  05  CF_COL_FIRSTNME       CHAR(12) VAR,
                  05  CF_COL_NUMBER         FIXED BIN(31)  INIT(0),
                  05  CF_COL_INDICATOR      FIXED BIN(31)  INIT(0),
                  05  CF_COL_OUTLEN         FIXED BIN(31);
          DCL
              01  LANG_FIELDS               STATIC,
                  05  LF_LANG               CHAR(36)
                      INIT ('SELECT FIRSTNME FROM SYBASE.SAMPLETB');
```

```
          DCL
              01  ERROR_MSG,
                  05  ERROR_TEXT             CHAR(50)       INIT(' '),
                  05  ERROR_LITERAL          CHAR(03)       INIT('RC='),
                  05  ERROR_RC               PIC '----9';
          DCL
              01  ERROR_MSG_STR              CHAR(58)
                                             DEFINED ERROR_MSG;
          DCL
              01  OUTPUT_MSG,
                  05  MESG                   CHAR(27)       INIT(' '),
                  05  NUM_COLS_OR_ROWS       PIC '99999';
          DCL
              01  OUTPUT_MSG_STR             CHAR(32)
                                             DEFINED OUTPUT_MSG;
          DCL
              01  INFO_MSG_STR               CHAR(80)       INIT(' ');
          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);
/*-------------------------------------------------------------------*/
/* COMMON WORK AREAS                                                 */
/*-------------------------------------------------------------------*/
          DCL
              01  MSG_FIELDS                 STATIC,
                  05  MSG_END_MSG            CHAR(25)
                      INIT ('All done processing rows.'),
                  05  MSG_NOT_RPC            CHAR(35)
                      INIT ('SYX4 must be begun via rpc request.'),
                  05  MSG_WRONG_NR_PARMS     CHAR(50)
                      INIT ('Number of parameters must be 2 or 3.'),
                  05  MSG_NOT_CHAR_PARM      CHAR(50)
                      INIT ('Parameter must be a CHARACTER type.'),
```

```
        05  MSG_NOT_INT4_PARM     CHAR(50)
            INIT ('Parameter must be a INTEGER type.'),
        05  MSG_CANCELED          CHAR(17)
            INIT ('Cancel requested.'),
        05  MSG_TDRCVPRM_FAIL     CHAR(16)
            INIT ('TDRCVPRM failed.');
DCL
    01  CICS_FIELDS,
        05  CICS_RESPONSE         FIXED BIN(31);
DCL
     01  MISC_FIELDS,
         05  BLANK                CHAR(01) INIT(' '),
         05  BLANK_13             CHAR(13) INIT(' '),
         05  DIAG_MSGS_INIT       BIT(1)   INIT('1'B),
         05  FALSE                BIT(1)   INIT('0'B),
         05  LCV                  FIXED BIN(31),
         05  TMP_TIME             CHAR(08) INIT(' '),
         05  TMP_DATE             CHAR(08) INIT(' '),
         05  TRUE                 BIT(1)   INIT('1'B),
         05  UTIME                FIXED DEC(15) INIT(0);

DCL
    01  DIAG_FIELDS,
        05  DF_MSGNO              FIXED BIN(31) INIT(1),
        05  DF_NUM_OF_MSGS        FIXED BIN(31) INIT(0);

DCL
    01 DISP_MSG,
       05 TEST_CASE               CHAR(09) INIT('SYCTSAX4 '),
       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
    01 DISP_DATE_HEADER,
       05 DATE_HDR                CHAR(06) INIT(' DATE '),
       05 DATE_DATA               CHAR(08),
       05 X4_HDR                  CHAR(54) ;

DCL
    01 DISP_TIME_HEADER,
```

```
            05 TIME_HDR                CHAR(06) INIT(' TIME '),
            05 TIME_DATA               CHAR(08);

        DCL
            01 DISP_SERVER,
            05 SERVER_HDR              CHAR(09) INIT(' SERVER: '),
            05 SERVER_DATA             CHAR(30),
            05 USER_HDR                CHAR(10) INIT(' USER-ID: '),
            05 USER_DATA               CHAR(08);

/*-------------------------------------------------------------------*/
/* 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_5         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);
/*-------------------------------------------------------------------*/
/* Begin program here                                                */
/*-------------------------------------------------------------------*/

X4_HDR = ' SYBASE PL/1 SAMPLE PROGRAM SYCTSAX4 SQL RESULT OUTPUT' ;

EXEC CICS ASKTIME ABSTIME(UTIME);
```

```
EXEC CICS FORMATTIME
          ABSTIME(UTIME)
          DATESEP('/')
          MMDDYY(TMP_DATE)
          TIME(TMP_TIME)
          TIMESEP ;

/*-------------------------------------------------------------------*/
/* intialize the TDS environment for a client                        */
/*-------------------------------------------------------------------*/

CALL TDINIT( DFHEIBLK,
             GWL_RC,
             CSL_CTX_HANDLE);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDINIT failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

/*---------------------------------------*/
/* accept a request from a remote client */
/*---------------------------------------*/

CALL TDACCEPT( GWL_TDPROC,
               GWL_RC,
               CSL_CTX_HANDLE,
               SNA_CONNECTION_NAME,
               SNA_SUBC);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDACCEPT failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

/*---------------------------------------*/
/* display program heading information   */
/*---------------------------------------*/
```

```
INFO_MSG_STR = BLANK ;
CALL SEND_INFO_MESSAGE;

INFO_MSG_STR = DATE_HDR  || TMP_DATE || X4_HDR ;
CALL SEND_INFO_MESSAGE;

INFO_MSG_STR = TIME_HDR  || TMP_TIME ;
CALL SEND_INFO_MESSAGE;

INFO_MSG_STR = BLANK ;
CALL SEND_INFO_MESSAGE;

/*-------------------------------------------------*/
/* determine how many parameters were sent with    */
/* the current RPC by the remote client or server */
/* .. better be two                                */
/*-------------------------------------------------*/

CALL TDNUMPRM( GWL_TDPROC,
               PF_NUM_OF_PARMS);

IF PF_NUM_OF_PARMS = 2 | PF_NUM_OF_PARMS = 3
THEN ;
ELSE DO;

  INFO_MSG_STR = MSG_WRONG_NR_PARMS;
  CALL SEND_INFO_MESSAGE;

  INFO_MSG_STR = BLANK;
  CALL SEND_INFO_MESSAGE;

  INFO_MSG_STR =
     'syntax is: SYX4 server-nm, user-id   OR' ;
  CALL SEND_INFO_MESSAGE;

  INFO_MSG_STR =
     '           SYX4 server-nm, user-id, password' ;
  CALL SEND_INFO_MESSAGE;

  GO TO ALL_DONE;
END;

/*-------------------------------------------------------------*/
/* retrieves parameter type, datatype, and length information */
/* about the 1st RPC parameter( server-name parameter )       */
/*-------------------------------------------------------------*/
```

```
PF_PARM_ID = 1;

CALL TDINFPRM( GWL_TDPROC,
               GWL_RC,
               PF_PARM_ID,
               PF_DATATYPE,
               PF_ACTUAL_DATA_LENGTH,
               PF_MAX_DATA_LENGTH,
               PF_PARM_STATUS,
               PF_PARM_NAME,
               PF_PARM_NAME_LENGTH,
               TDS_NULL);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDINFPRM for server-name parameter failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

IF PF_DATATYPE ^= TDSCHAR & PF_DATATYPE ^= TDSVARYCHAR THEN
DO;
  INFO_MSG_STR = 'server-name datatype must be TDSCHAR';
  CALL SEND_INFO_MESSAGE;
  GO TO ALL_DONE;
END;

/*-------------------------------------------------------------*/
/* retrieves the data from an RPC parameter sent by a remote  */
/* client                                                      */
/*-------------------------------------------------------------*/

CALL TDRCVPRM( GWL_TDPROC,
               GWL_RC,
               PF_PARM_ID,
               PF_SERVER,
               TDSCHAR,
               STG(PF_SERVER),
               PF_ACTUAL_DATA_LENGTH);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDRCVPRM failed.';
  ERROR_RC   = GWL_RC;
```

```
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

PF_SERVER_LEN = PF_ACTUAL_DATA_LENGTH ;

/*-------------------------------------------------------------*/
/* retrieves parameter type, datatype, and length information */
/* about the 2nd RPC parameter( user-id parameter )           */
/*-------------------------------------------------------------*/

PF_PARM_ID = 2;

CALL TDINFPRM( GWL_TDPROC,
               GWL_RC,
               PF_PARM_ID,
               PF_DATATYPE,
               PF_ACTUAL_DATA_LENGTH,
               PF_MAX_DATA_LENGTH,
               PF_PARM_STATUS,
               PF_PARM_NAME,
               PF_PARM_NAME_LENGTH,
               TDS_NULL);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDINFPRM for user-id parameter failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

IF PF_DATATYPE ^= TDSCHAR & PF_DATATYPE ^= TDSVARYCHAR THEN
DO;
  INFO_MSG_STR = 'user-id datatype must be TDSCHAR';
  CALL SEND_INFO_MESSAGE;
  GO TO ALL_DONE;
END;

/*-------------------------------------------------------------*/
/* retrieves the data from an RPC parameter sent by a remote  */
/* client                                                      */
/*-------------------------------------------------------------*/

CALL TDRCVPRM( GWL_TDPROC,
               GWL_RC,
```

```
                PF_PARM_ID,
                PF_USER,
                TDSCHAR,
                STG(PF_USER),
                PF_ACTUAL_DATA_LENGTH);

 IF GWL_RC ^= TDS_OK THEN
 DO;
   ERROR_TEXT = 'TDRCVPRM for user-id failed.';
   ERROR_RC   = GWL_RC;
   CALL SEND_ERROR_MESSAGE;
   GO TO ALL_DONE;
 END;

 PF_USER_LEN = PF_ACTUAL_DATA_LENGTH ;



/*-------------------------------------------------------------*/
 /* retrieves parameter type, datatype, and length information */
 /* about the 3rd RPC parameter( password parameter )          */
 /*-------------------------------------------------------------*/
 IF PF_NUM_OF_PARMS = 3 THEN
 DO ;
   PF_PARM_ID = 3;

   CALL TDINFPRM( GWL_TDPROC,
                  GWL_RC,
                  PF_PARM_ID,
                  PF_DATATYPE,
                  PF_ACTUAL_DATA_LENGTH,
                  PF_MAX_DATA_LENGTH,
                  PF_PARM_STATUS,
                  PF_PARM_NAME,
                  PF_PARM_NAME_LENGTH,
                  TDS_NULL);

   IF GWL_RC ^= TDS_OK THEN
   DO;
     ERROR_TEXT = 'TDINFPRM for password parameter failed.';
     ERROR_RC   = GWL_RC;
     CALL SEND_ERROR_MESSAGE;
     GO TO ALL_DONE;
   END;

   IF PF_DATATYPE ^= TDSCHAR & PF_DATATYPE ^= TDSVARYCHAR THEN
```

```
  DO;
    INFO_MSG_STR = 'password datatype must be TDSCHAR';
    CALL SEND_INFO_MESSAGE;
    GO TO ALL_DONE;
  END;

/*-------------------------------------------------------------*/
/* retrieves the data from an RPC parameter sent by a remote  */
/* client                                                      */
/*-------------------------------------------------------------*/

  CALL TDRCVPRM( GWL_TDPROC,
                 GWL_RC,
                 PF_PARM_ID,
                 PF_PWD,
                 TDSCHAR,
                 STG(PF_PWD),
                 PF_ACTUAL_DATA_LENGTH);

  IF GWL_RC ^= TDS_OK THEN
  DO;
    ERROR_TEXT = 'TDRCVPRM for password failed.';
    ERROR_RC   = GWL_RC;
    CALL SEND_ERROR_MESSAGE;
    GO TO ALL_DONE;
  END;

  PF_PWD_LEN = PF_ACTUAL_DATA_LENGTH ;
END ;
ELSE DO ;
  PF_PWD     = BLANK ;
  PF_PWD_LEN = 0 ;
END ;
/*--------------------------------------*/
/* display server and user-id heading    */
/*--------------------------------------*/

INFO_MSG_STR = SERVER_HDR || PF_SERVER ||
               USER_HDR   || PF_USER ;
CALL SEND_INFO_MESSAGE;

INFO_MSG_STR = BLANK ;
CALL SEND_INFO_MESSAGE;


INFO_MSG_STR = BLANK ;
```

```
CALL SEND_INFO_MESSAGE;

/*-------------------------------------------------------------*/
/* describe the 1st column in a result row and the mainframe */
/* server program variable where it is stored                */
/*-------------------------------------------------------------*/

DF_COLUMN_NUMBER = 1;
DF_HOST_VARIABLE_TYPE   = TDSVARYCHAR;
DF_HOST_VARIABLE_MAXLEN = STG(CF_COL_FIRSTNME) - 2;
DF_HOST_VARIABLE_NAME   = ADDR(CF_COL_FIRSTNME);
DF_NULL_INDICATOR_VAR   = TDS_ZERO;    /* not null */
DF_NULLS_ALLOWED        = TDS_FALSE;
DF_COLUMN_TYPE          = TDSVARYCHAR;
DF_COLUMN_MAXLEN        = STG(CF_COL_FIRSTNME) - 2;
DF_COLUMN_NAME          = 'FIRST NAME';
DF_COLUMN_NAME_LEN      = 10;          /* length of 'FIRST NAME' */

CALL TDESCRIB( GWL_TDPROC,
               GWL_RC,
               DF_COLUMN_NUMBER,
               DF_HOST_VARIABLE_TYPE,
               DF_HOST_VARIABLE_MAXLEN,
               CF_COL_FIRSTNME,
               DF_NULL_INDICATOR_VAR,
               DF_NULLS_ALLOWED,
               DF_COLUMN_TYPE,
               DF_COLUMN_MAXLEN,
               DF_COLUMN_NAME,
               DF_COLUMN_NAME_LEN);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDESCRIB failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
  GO TO ALL_DONE;
END;

/*-------------------------------------------------------------*/
/* Open Client calls                                           */
/*-------------------------------------------------------------*/

CALL OC_INIT;

CALL OC_CONNECT;
```

```
CALL OC_SEND_LANG;

CALL OC_PROCESS_RESULTS;

CALL OC_ALL_DONE;

ALL_DONE:

/*-------------------------------------------------------------*/
/* send a results completion indication to the client          */
/*-------------------------------------------------------------*/

CALL TDSNDDON( GWL_TDPROC,
               GWL_RC,
               TDS_DONE_FINAL,
               TDS_NULL,
               TDS_ZERO,
               TDS_ENDRPC);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDSNDDON failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
END;

/*-------------------------------------------------------------*/
/* free up a previously allocated GWL_TDPROC structure after   */
/* returning results to a client                               */
/*-------------------------------------------------------------*/

CALL TDFREE( GWL_TDPROC,
             GWL_RC);

IF GWL_RC ^= TDS_OK THEN
DO;
  ERROR_TEXT = 'TDFREE failed.';
  ERROR_RC   = GWL_RC;
  CALL SEND_ERROR_MESSAGE;
END;

EXEC CICS RETURN;

/*-------------------------------------------------------------------*/
/* Subroutine to initialize the Client-Library                       */
```

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

OC_INIT: PROC;

/*--------------------------------------------------------------------*/
/* initialize the Client-Library                                      */
/*--------------------------------------------------------------------*/

        CALL CTBINIT( CSL_CTX_HANDLE,
                      CSL_RC,
                      CS_VERSION_46);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBINIT failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

END OC_INIT;

/*--------------------------------------------------------------------*/
/* Subroutine to allocate connect handler, alter properties for       */
/* user-id and password, set up retrieval of all Open Client          */
/* messages, and open connection to the server                        */
/*--------------------------------------------------------------------*/

OC_CONNECT: PROC;

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

        CALL CTBCONALLOC( CSL_CTX_HANDLE,
                          CSL_RC,
                          CSL_CON_HANDLE);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCONALLOC failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

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

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

        PF_STRLEN = STG(PF_USER);

        CALL CTBCONPROPS( CSL_CON_HANDLE,
                          CSL_RC,
                          CS_SET,
                          CS_USERNAME,
                          PF_USER,
                          PF_USER_LEN,
                          CS_FALSE,
                          CS_UNUSED);

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

        PF_STRLEN = 0;

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

        CALL CTBCONPROPS( CSL_CON_HANDLE,
                          CSL_RC,
                          CS_SET,
                          CS_PASSWORD,
                          PF_PWD,
                          PF_PWD_LEN,
                          CS_FALSE,
                          CS_UNUSED);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCONPROPS for password failed.';
            CALL ERROR_OUT;
            GO TO 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.';
            CALL ERROR_OUT;
            GO TO 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.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END ;

/*-------------------------------------------------------------*/
/* open connection to the server                               */
/*-------------------------------------------------------------*/

        CALL CTBCONNECT( CSL_CON_HANDLE,
                         CSL_RC,
                         PF_SERVER,
                         PF_SERVER_LEN,
                         CS_FALSE);
```

```
        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCONNECT failed' ;
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

END OC_CONNECT;

/*-------------------------------------------------------------------*/
/* Subroutine to allocate command handler, prepare and send the     */
/* language request                                                  */
/*-------------------------------------------------------------------*/

OC_SEND_LANG: PROC;

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

        CALL CTBCMDALLOC( CSL_CON_HANDLE,
                          CSL_RC,
                          CSL_CMD_HANDLE);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCMDALLOC failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

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

        PF_STRLEN = STG(LF_LANG);

        CALL CTBCOMMAND( CSL_CMD_HANDLE,
                         CSL_RC,
                         CS_LANG_CMD,
                         LF_LANG,
                         PF_STRLEN,
                         CS_UNUSED);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
```

```
            MSGSTR = 'CTBCOMMAND failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

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

        CALL CTBSEND( CSL_CMD_HANDLE,
                      CSL_RC);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBSEND failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

END OC_SEND_LANG;

/*-------------------------------------------------------------------*/
/* Subroutine to process the result                                  */
/*-------------------------------------------------------------------*/

OC_PROCESS_RESULTS: PROC;

        DORESULTS: DO WHILE(1=1);

/*-------------------------------------------------------------*/
/* set up the results data                                     */
/*-------------------------------------------------------------*/

            CALL CTBRESULTS( CSL_CMD_HANDLE,
                             CSL_RC,
                             RF_TYPE);

            IF CSL_RC ^= CS_SUCCEED
              THEN
                LEAVE DORESULTS;

/*-------------------------------------------------------------*/
/* 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().         */
/*-------------------------------------------------------------*/
```

```
IF RF_TYPE = CS_ROW_RESULT THEN
DO;

    /*---------------------------------*/
    /* 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 */

    /*------------------------*/
    /* bind the first column */
    /*------------------------*/

    CF_COL_NUMBER = 1;

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

    IF CSL_RC ^= CS_SUCCEED THEN
    DO;
        MSGSTR = 'CTBBIND first name failed.';
        CALL ERROR_OUT;
        GO TO ALL_DONE;
    END;

    /*-----------------------------------------*/
    /* loop to fetch all rows in this result set */
    /*-----------------------------------------*/

    DOFETCHROW: DO WHILE(1=1);

        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
                        LEAVE DOFETCHROW;

                    FF_ROW_NUM = FF_ROW_NUM + 1;

                /*-------------------------------------------*/
                /* send a row of data back to the requesting */
                /* client                                    */
                /*-------------------------------------------*/

                    CALL TDSNDROW( GWL_TDPROC,
                                   GWL_RC);

                    IF GWL_RC ^= TDS_OK THEN
                    DO;
                      INFO_MSG_STR = MSG_CANCELED;
                      CALL SEND_INFO_MESSAGE;
                    END;

                END; /* end of DOFETCHROW */

/*-----------------------------------------------------*/
/* We're done processing rows. Let's check the final  */
/* return value of ctbfetch.                          */
/*-----------------------------------------------------*/

                IF CSL_RC = CS_CANCELLED THEN
                DO;
                    ERROR_TEXT = MSG_CANCELED;
                    ERROR_RC   = 0;
                    CALL SEND_ERROR_MESSAGE;
                END;

                ELSE IF CSL_RC = CS_END_DATA THEN
                DO;
                    INFO_MSG_STR = BLANK;
                    CALL SEND_INFO_MESSAGE;
                    INFO_MSG_STR = MSG_END_MSG;
                    CALL SEND_INFO_MESSAGE;
                END;
```

```
                ELSE IF CSL_RC = CS_SUCCEED THEN
                DO;
                    ERROR_TEXT = 'CTBFETCH unexpected return code';
                    ERROR_RC   = CSL_RC;
                    CALL SEND_ERROR_MESSAGE;
                END;

                ELSE DO;
                    ERROR_TEXT = 'CTBFETCH failed.';
                    ERROR_RC   = CSL_RC;
                    CALL SEND_ERROR_MESSAGE;
                END;

            END; /* end rftype = cs_row_result */

/*-------------------------------------------------------------*/
/* don't care about status                                     */
/*-------------------------------------------------------------*/

            ELSE IF RF_TYPE = CS_STATUS_RESULT THEN
            DO;
                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.';
                    CALL ERROR_OUT;
                END;
            END;

/*-------------------------------------------------------------*/
/* print an error message if the server encountered an error   */
/* while executing the request                                 */
/*-------------------------------------------------------------*/

            ELSE IF RF_TYPE = CS_CMD_FAIL THEN
            DO;
              MSGSTR = 'CTBRESUL returned CS_CMD-FAIL restype' ;
              CALL ERROR_OUT;

              INFO_MSG_STR = 'bad user-id or password' ;
```

```
              CALL SEND_INFO_MESSAGE;

              INFO_MSG_STR = BLANK ;
              CALL SEND_INFO_MESSAGE;
            END;

/*-------------------------------------------------------*/
/* this means no rows were returned                      */
/*-------------------------------------------------------*/

            ELSE IF RF_TYPE = CS_SUCCEED THEN
            DO;
            END;

/*-------------------------------------------------------*/
/* done with one result set, let's go on to the next     */
/*-------------------------------------------------------*/

            ELSE IF RF_TYPE = CS_CMD_DONE THEN
            DO;
            END;

/*-------------------------------------------------------*/
/* the server encountered an error while processing our  */
/* command                                               */
/*-------------------------------------------------------*/

            ELSE IF RF_TYPE = CS_CMD_FAIL THEN
            DO;
            END;

/*-------------------------------------------------------*/
/* we got something unexpected                           */
/*-------------------------------------------------------*/

            ELSE
            DO;
            END;
        END; /* end of DORESULTS */

/*-------------------------------------------------------------*/
/* We're done processing results. Let's check the return value */
/* of CTBRESULTS() to see if everything went ok.               */
/*-------------------------------------------------------------*/

        IF CSL_RC = CS_END_RESULTS THEN
```

```
        DO;
        END;

/*-------------------------------------------------------------*/
/* something terrible happened                                 */
/*-------------------------------------------------------------*/

        ELSE IF CSL_RC = CS_FAIL THEN
        DO;
            MSGSTR = 'CTBRESULTS failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

/*-------------------------------------------------------------*/
/* we got an unexpected return value                           */
/*-------------------------------------------------------------*/

        ELSE
        DO;
            MSGSTR = 'CTBRESULTS failed.';
            CALL ERROR_OUT;
            GO TO ALL_DONE;
        END;

END OC_PROCESS_RESULTS;

/*-------------------------------------------------------------------*/
/* Subroutine to drop the command handler, to close the server       */
/* connection, to drop the connection handler and exit               */
/*-------------------------------------------------------------------*/

OC_ALL_DONE: PROC;

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

        CALL CTBCMDDROP( CSL_CMD_HANDLE,
                         CSL_RC);

        IF CSL_RC ^= CS_SUCCEED 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_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCLOSE failed.';
            CALL ERROR_OUT;
        END;

/*--------------------------------------------------------------------*/
/* de-allocate the connection handler                                 */
/*--------------------------------------------------------------------*/

        CALL CTBCONDROP( CSL_CON_HANDLE,
                         CSL_RC);

        IF CSL_RC ^= CS_SUCCEED THEN
        DO;
            MSGSTR = 'CTBCONDROP failed.';
            CALL ERROR_OUT;
        END;

END OC_ALL_DONE;

/*--------------------------------------------------------------------*/
/* Subroutine to send an error message to the client                  */
/*--------------------------------------------------------------------*/

SEND_ERROR_MESSAGE: PROC;

    SF_MESSAGE_TYPE       = TDS_ERROR_MSG;
    SF_MESSAGE_NUMBER     = 0;
    SF_SEVERITY           = 10; /* TDS_ERROR_SEV */
    SF_ERROR_STATE        = 0;
    SF_LINE_ID            = 0;
    SF_TRANSACTION_ID_LEN = STG(SF_TRANSACTION_ID);
    SF_MESSAGE_TEXT       = ERROR_MSG_STR;
    SF_MESSAGE_LENGTH     = STG(SF_MESSAGE_TEXT);

    CALL TDSNDMSG( GWL_TDPROC,
```

```
                    GWL_RC,
                    SF_MESSAGE_TYPE,        /* msg type      */
                    SF_MESSAGE_NUMBER,      /* return_code   */
                    SF_SEVERITY,            /* TDS_ERROR_SEV */
                    SF_ERROR_STATE,         /* error state   */
                    SF_LINE_ID,             /* line #        */
                    SF_TRANSACTION_ID,      /* transaction id*/
                    SF_TRANSACTION_ID_LEN,  /* tid len       */
                    SF_MESSAGE_TEXT,        /* msg           */
                    SF_MESSAGE_LENGTH);     /* msg len       */

END SEND_ERROR_MESSAGE;

/*--------------------------------------------------------------------*/
/* Subroutine to send an information message to the client           */
/*--------------------------------------------------------------------*/

SEND_INFO_MESSAGE: PROC;

    SF_MESSAGE_TYPE       = TDS_INFO_MSG;
    SF_MESSAGE_NUMBER     = 0;
    SF_SEVERITY           = 0; /* TDS_INFO_SEV */
    SF_ERROR_STATE        = 0;
    SF_LINE_ID            = 0;
    SF_TRANSACTION_ID_LEN = STG(SF_TRANSACTION_ID);
    SF_MESSAGE_TEXT       = INFO_MSG_STR;
    SF_MESSAGE_LENGTH     = STG(SF_MESSAGE_TEXT);

    CALL TDSNDMSG( GWL_TDPROC,
                   GWL_RC,
                   SF_MESSAGE_TYPE,        /* msg type      */
                   SF_MESSAGE_NUMBER,      /* return_code   */
                   SF_SEVERITY,            /* TDS_ERROR_SEV */
                   SF_ERROR_STATE,         /* error state   */
                   SF_LINE_ID,             /* line #        */
                   SF_TRANSACTION_ID,      /* transaction id*/
                   SF_TRANSACTION_ID_LEN,  /* tid len       */
                   SF_MESSAGE_TEXT,        /* msg           */
                   SF_MESSAGE_LENGTH);     /* msg len       */

END SEND_INFO_MESSAGE;

/*--------------------------------------------------------------------*/
/*                                                                    */
/* Subroutine to print output messages.                               */
/*                                                                    */
```

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

 ERROR_OUT: PROC;

         IF DIAG_MSGS_INIT
           THEN
             CALL GET_DIAG_MESSAGES ;



/*-------------------------------------------------------------------*/
 /* display error messages                                           */
 /*-------------------------------------------------------------------*/

         SAMP_RC   = CSL_RC;
         REST_TYPE = RF_TYPE ;

         INFO_MSG_STR = BLANK ;
         CALL SEND_INFO_MESSAGE;

         INFO_MSG_STR = TEST_CASE || SAMP_LIT  || SAMP_RC ||
                        REST_LIT  || REST_TYPE || '   '   ||
                        MSGSTR ;
         CALL SEND_INFO_MESSAGE;

         INFO_MSG_STR = BLANK ;
         CALL SEND_INFO_MESSAGE;

         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_INIT = 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 ;
            ERROR_TEXT = 'CTBDIAG CS_STATUS CLIENTMSG_TYPE failed';
            ERROR_RC   = CSL_RC;
            CALL SEND_ERROR_MESSAGE;
            GO TO 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 ;
            ERROR_TEXT = 'CTBDIAG CS_STATUS SERVERMSG_TYPE failed' ;
            ERROR_RC   = CSL_RC;
```

```
            CALL SEND_ERROR_MESSAGE;
            GO TO 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 ;

        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 ;
            ERROR_TEXT = 'CTBDIAG CS_GET CS_CLIENTMSG_TYPE FAILED' ;
            ERROR_RC   = CSL_RC;
            CALL SEND_ERROR_MESSAGE;
            GO TO ALL_DONE;
        END ;

/*--------------------------------------------------------------------*/
/* display message text                                               */
/*--------------------------------------------------------------------*/

        INFO_MSG_STR = 'Client Message:' ;
        CALL SEND_INFO_MESSAGE;
        INFO_MSG_STR = BLANK ;
        CALL SEND_INFO_MESSAGE;
```

```
CM_SEVERITY_DATA = CM_SEVERITY ;
CM_STATUS_DATA   = CM_STATUS ;
INFO_MSG_STR     = CM_SEVERITY_HDR || CM_SEVERITY_DATA ||
                   CM_STATUS_HDR   || CM_STATUS_DATA ;
CALL SEND_INFO_MESSAGE;

CM_OC_MSGNO_DATA = CM_MSGNO ;
INFO_MSG_STR     = CM_OC_MSGNO_HDR || CM_OC_MSGNO_DATA ;
CALL SEND_INFO_MESSAGE;

IF CM_MSGNO ^= 0 THEN
DO ;
    CM_OC_MSG_DATA     = SUBSTR( CM_TEXT, 1, 66 ) ;
    INFO_MSG_STR       = CM_OC_MSG_HDR || CM_OC_MSG_DATA ;
    CALL SEND_INFO_MESSAGE;

    IF CM_TEXT_LEN > 66 THEN
    DO ;
        CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 67, 66 ) ;
        INFO_MSG_STR     = BLANK_13 || CM_OC_MSG_DATA_X ;
        CALL SEND_INFO_MESSAGE;

        IF CM_TEXT_LEN > 132 THEN
        DO ;
            CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 133, 66 ) ;
            INFO_MSG_STR     = BLANK_13 ||
                               CM_OC_MSG_DATA_X ;
            CALL SEND_INFO_MESSAGE;

            IF CM_TEXT_LEN > 198 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_TEXT, 199 ) ;
                INFO_MSG_STR     = BLANK_13 ||
                                   CM_OC_MSG_DATA_X ;
                CALL SEND_INFO_MESSAGE;

            END ;
        END ;
    END ;
END ;
ELSE DO ;
  INFO_MSG_STR = '  OC MsgTx: No Message' ;
  CALL SEND_INFO_MESSAGE;
END ;

CM_OS_MSGNO_DATA = CM_OS_MSGNO ;
```

```
        INFO_MSG_STR     = '  OS MsgNo:  ' || CM_OS_MSGNO_DATA ;
        CALL SEND_INFO_MESSAGE;

        IF CM_OS_MSGNO ^= 0 THEN
        DO ;
            CM_OS_MSG_DATA     = SUBSTR( CM_OS_MSGTXT, 1, 66 ) ;
            INFO_MSG_STR       = CM_OS_MSG_HDR_5 ||
                                 CM_OS_MSG_DATA ;
            CALL SEND_INFO_MESSAGE;

            IF CM_OS_MSGTEXT_LEN > 66 THEN
            DO ;
                CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT, 67, 66 ) ;
                INFO_MSG_STR     = BLANK_13 || CM_OC_MSG_DATA_X ;
                CALL SEND_INFO_MESSAGE;

                IF CM_OS_MSGTEXT_LEN > 132 THEN
                DO ;
                    CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,133,66 );
                    INFO_MSG_STR     = BLANK_13 || CM_OC_MSG_DATA_X ;
                    CALL SEND_INFO_MESSAGE;

                    IF CM_OS_MSGTEXT_LEN > 198 THEN
                    DO ;
                        CM_OC_MSG_DATA_X = SUBSTR( CM_OS_MSGTXT,199 );
                        INFO_MSG_STR     = BLANK_13 ||
                                           CM_OC_MSG_DATA_X ;
                        CALL SEND_INFO_MESSAGE;
                    END ;
                END ;
            END ;
        END ;
        ELSE DO ;
          INFO_MSG_STR = '  OS MsgTx:  No Message!' ;
          CALL SEND_INFO_MESSAGE;
        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 ;
            ERROR_TEXT = 'CTBDIAG CS_GET CS_SERVERMSG_TYPE failed' ;
            ERROR_RC   = CSL_RC;
            CALL SEND_ERROR_MESSAGE;
            GO TO 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 ) ;

        INFO_MSG_STR = 'Server Message:' ;
        CALL SEND_INFO_MESSAGE;
        INFO_MSG_STR = BLANK ;
        CALL SEND_INFO_MESSAGE;

        INFO_MSG_STR = SM_MSG_NO_HDR   || SM_MSG_NO_DATA ||
```

```
                    SM_SEVERITY_HDR || SM_SEVERITY_DATA ||
                    SM_STATE_HDR    || SM_STATE_DATA ;
          CALL SEND_INFO_MESSAGE;

          INFO_MSG_STR = SM_LINE_NO_HDR  || SM_LINE_NO_DATA ||
                         SM_STATUS_HDR   || SM_STATUS_DATA ;
          CALL SEND_INFO_MESSAGE;

          INFO_MSG_STR = SM_SVRNAME_HDR  || SM_SVRNAME_DATA ;
          CALL SEND_INFO_MESSAGE;

          INFO_MSG_STR = SM_PROC_ID_HDR  || SM_PROC_ID_DATA ;
          CALL SEND_INFO_MESSAGE;

          INFO_MSG_STR = SM_MSG_HDR      || SM_MSG_DATA ;
          CALL SEND_INFO_MESSAGE;

          IF SM_TEXT_LEN > 66 THEN
            DO ;
              SM_MSG_DATA_X = SUBSTR( SM_TEXT, 67, 66 ) ;
              INFO_MSG_STR  = BLANK_13 || SM_MSG_DATA_X ;
              CALL SEND_INFO_MESSAGE;

              IF SM_TEXT_LEN > 132 THEN
                DO ;
                  SM_MSG_DATA_X = SUBSTR( SM_TEXT, 133, 66 ) ;
                  INFO_MSG_STR  = BLANK_13 || SM_MSG_DATA_X ;
                  CALL SEND_INFO_MESSAGE;

                  IF SM_TEXT_LEN > 198 THEN
                    DO ;
                      SM_MSG_DATA_X = SUBSTR( SM_TEXT, 198 ) ;
                      INFO_MSG_STR = BLANK_13 || SM_MSG_DATA_X ;
                      CALL SEND_INFO_MESSAGE;
                    END ;
                END ;
            END ;

END RETRIEVE_SERVER_MSGS ;

END  SYCTSAX4;
     *@(#) syctsax5.cobol 1.2 4/18/96
```

APPENDIX G

Sample Tracing and Accounting Program

This appendix contains a sample host server application program that a system programmer can use to perform host-based tracing and accounting functions.

Note This program is not included on the Open ServerConnect tape.

The purpose of this sample program is to demonstrate the use of all Gateway-Library tracing and accounting functions. In some cases, one Gateway-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.

Sample program SYCPSAS2

This program uses the Gateway-Library system programmer calls to do tracing and accounting at the host.

```
SYSAMP1: PROC OPTIONS(MAIN REENTRANT);
 /****** SYCPSAS1 - RPC REQUEST APPLICATION - PL/1 - CICS ********/
 /*                                                               */
 /*  TRANID:         SYS1                                         */
 /*  PROGRAM:        SYCPSAS1                                     */
 /*  PLAN NAME:      n/a                                          */
 /*  FILES:          n/a                                          */
 /*  TABLES:         n/a                                          */
 /*                                                               */
 /*  This program is executed via a client RPC request from sample*/
 /*  dblib program 'SYS1'.  The purpose of the program is primarily*/
 /*  to demonstrate system programmer Server Library calls.       */
 /*                                                               */
 /*  Server Library calls:                                        */
 /*    TDACCEPT      accept request from client                   */
```

```
  /*    TDESCRIB      describe a column                                 */
  /*    TDFREE        free TDPROC structure                             */
  /*    TDINFACT      get accounting info                               */
  /*    TDINFLOG      get logging info                                  */
  /*    TDINFSPT      get specific tracing info                         */
  /*    TDINIT        establish environment                             */
  /*    TDLSTSPT      get list of active specific trace tran ids        */
  /*    TDRCVPRM      retrieve rpc parameter from client                */
  /*    TDRESULT      describe next communication                       */
  /*    TDSETACT      set accounting                                    */
  /*    TDSETLOG      set logging                                       */
  /*    TDSETSPT      set specific tracing                              */
  /*    TDSNDDON      send results-completion to client                 */
  /*    TDSNDMSG      send message to client                            */
  /*    TDSNDROW      send row to client                                */
  /*    TDSTATUS      get status information                            */
  /*    TDWRTLOG      write user log record                             */
  /*                                                                    */
  /********************************************************************/

  /********************************************************************/
  /*      GATEWAY-LIBRARY PL/1 COPY BOOK                              */
  /********************************************************************/
          %INCLUDE SYGWPLI;
/*------------------------------------------------------------------*/
/*      SERVER LIB ROUTINES DECLARATIONS                            */
/*------------------------------------------------------------------*/
          DCL
              TDACCEPT ENTRY OPTIONS(INTER ASSEMBLER),
              TDESCRIB ENTRY OPTIONS(INTER ASSEMBLER),
              TDFREE   ENTRY OPTIONS(INTER ASSEMBLER),
              TDINFACT ENTRY OPTIONS(INTER ASSEMBLER),
              TDINFLOG ENTRY OPTIONS(INTER ASSEMBLER),
              TDINFSPT ENTRY OPTIONS(INTER ASSEMBLER),
              TDINIT   ENTRY OPTIONS(INTER ASSEMBLER),
              TDLSTSPT ENTRY OPTIONS(INTER ASSEMBLER),
              TDRCVPRM ENTRY OPTIONS(INTER ASSEMBLER),
              TDRESULT ENTRY OPTIONS(INTER ASSEMBLER),
              TDSETACT ENTRY OPTIONS(INTER ASSEMBLER),
              TDSETLOG ENTRY OPTIONS(INTER ASSEMBLER),
              TDSETSPT ENTRY OPTIONS(INTER ASSEMBLER),
              TDSNDDON ENTRY OPTIONS(INTER ASSEMBLER),
              TDSNDMSG ENTRY OPTIONS(INTER ASSEMBLER),
              TDSNDROW ENTRY OPTIONS(INTER ASSEMBLER),
              TDSTATUS ENTRY OPTIONS(INTER ASSEMBLER),
              TDWRTLOG ENTRY OPTIONS(INTER ASSEMBLER);
```

```
/*-------------------------------------------------------------*/
/*      BUILT IN FUNCTIONS DECLARATIONS                        */
/*-------------------------------------------------------------*/
         DCL
             ADDR      BUILTIN,
             STG       BUILTIN,
             STRING    BUILTIN;
/*-------------------------------------------------------------*/
/*      WORK AREAS                                             */
/*-------------------------------------------------------------*/
         DCL
             01  GW_LIB_MISC_FIELDS,
                 05  GWL_PROC                  PTR,
                 05  GWL_INIT_HANDLE           PTR,
                 05  GWL_INFACT_STATUS         FIXED BIN(31),
                 05  GWL_INFACT_FILENAME       CHAR(08),
                 05  GWL_INFACT_RECORDS        FIXED BIN(31),
                 05  GWL_INFLOG_GLOBAL         FIXED BIN(31),
                 05  GWL_INFLOG_API            FIXED BIN(31),
                 05  GWL_INFLOG_HEADER         FIXED BIN(31),
                 05  GWL_INFLOG_DATA           FIXED BIN(31),
                 05  GWL_INFLOG_TRACEID        FIXED BIN(31),
                 05  GWL_INFLOG_FILENAME       CHAR(08),
                 05  GWL_INFLOG_RECORDS        FIXED BIN(31),
                 05  GWL_INFSPT_ENTRY          FIXED BIN(31),
                 05  GWL_INFSPT_STATUS         FIXED BIN(31),
                 05  GWL_INFSPT_OPTIONS        FIXED BIN(31),
                 05  GWL_INFSPT_TRANID         CHAR(04),
                 05  GWL_INFSPT_TRANID_L       FIXED BIN(31),
                 05  GWL_LSTSPT_LIST(8)        CHAR(08),
                 05  GWL_RC                    FIXED BIN(31),
                 05  GWL_RCVPRM_ID             FIXED BIN(31) INIT(1),
                 05  GWL_RCVPRM_MAX_DATA_L     FIXED BIN(31) INIT(2),
                 05  GWL_RCVPRM_DATA_L         FIXED BIN(31) INIT(2),
                 05  GWL_SETSPT_ENTRY          FIXED BIN(31),
                 05  GWL_SETSPT_OPTIONS        FIXED BIN(31),
                 05  GWL_STATUS_NR             FIXED BIN(31),
                 05  GWL_STATUS_DONE           FIXED BIN(31),
                 05  GWL_STATUS_COUNT          FIXED BIN(31),
                 05  GWL_STATUS_COMM           FIXED BIN(31),
                 05  GWL_STATUS_RETURN_CODE    FIXED BIN(31),
                 05  GWL_STATUS_SUBCODE        FIXED BIN(31),
                 05  GWL_WRTLOG_MSG_L          FIXED BIN(31) INIT(34),
                 05  GWL_WRTLOG_MSG            CHAR(34)
                     INIT('TEST MESSAGE FROM SYS1 TRANSACTION');
```

```
          DCL
              01  PARM_FIELDS,
                  05  PARM_REQUEST              CHAR(02);
DCL
              01  PARM_FIELDS_VALUES            STATIC,
                  05  PARM_REQUEST_INFACT       CHAR(02) INIT('IA'),
                  05  PARM_REQUEST_INFLOG       CHAR(02) INIT('IL'),
                  05  PARM_REQUEST_LSTSPT       CHAR(02) INIT('IS'),
                  05  PARM_REQUEST_SETACT_ON    CHAR(02) INIT('YA'),
                  05  PARM_REQUEST_SETACT_OFF   CHAR(02) INIT('NA'),
                  05  PARM_REQUEST_SETLOG_ON    CHAR(02) INIT('YL'),
                  05  PARM_REQUEST_SETLOG_OFF   CHAR(02) INIT('NL'),
                  05  PARM_REQUEST_SETSPT_ON    CHAR(02) INIT('YS'),
                  05  PARM_REQUEST_SETSPT_OFF   CHAR(02) INIT('NS'),
                  05  PARM_REQUEST_WRTLOG       CHAR(02) INIT('WL');
DCL
              01  SNA_FIELDS,
                  05  SNA_SUBC                  FIXED BIN(31),
                  05  SNA_CONNECTION_NAME       CHAR(08) INIT(' ');
DCL
              01  COLUMN_NAME_FIELDS            STATIC,
                  05  CN_INFACT_STATUS          CHAR(13)
                                                INIT('ACT_STATUS'),
                  05  CN_INFACT_FILENAME        CHAR(12)
                                                INIT('ACT_FILENAME'),
                  05  CN_INFACT_RECORDS         CHAR(11)
                                                INIT('ACT_RECORDS'),
                  05  CN_INFLOG_GLOBAL          CHAR(10)
                                                INIT('LOG_GLOBAL'),
                  05  CN_INFLOG_API             CHAR(07)
                                                INIT('LOG_API'),
                  05  CN_INFLOG_HEADER          CHAR(10)
                                                INIT('LOG_HEADER'),
                  05  CN_INFLOG_DATA            CHAR(08)
                                                INIT('LOG_DATA'),
                  05  CN_INFLOG_TRACEID         CHAR(11)
                                                INIT('LOG_TRACEID'),
                  05  CN_INFLOG_FILENAME        CHAR(12)
                                                INIT('LOG_FILENAME'),
                  05  CN_INFLOG_RECORDS         CHAR(11)
                                                INIT('LOG_RECORDS'),
                  05  CN_LSTSPT_TRANID          CHAR(06)
                                                INIT('TRANID');
DCL
              01  COUNTER_FIELDS,
                  05  CTR_COLUMN                FIXED BIN(31) INIT(0),
```

```
                05  CTR_ROWS                  FIXED BIN(31) INIT(0);
DCL
             01  WORK_FIELDS,
                 05  WRKLEN1                  FIXED BIN(31),
                 05  WRKLEN2                  FIXED BIN(31),
                 05  WRK_DONE_STATUS          FIXED BIN(31),
                 05  WRK_RPC                  CHAR(04) INIT('SYS1'),
                 05  WRK_LSTSPT_SS            FIXED BIN(15),
                 05  WRK_TRANID               CHAR(08);
DCL
            01  MESSAGE_FIELDS,
                05   MSG_TYPE                 FIXED BIN(31),
                05   MSG_SEVERITY_ERROR       FIXED BIN(31) INIT(11),
                05   MSG_NR_ERROR             FIXED BIN(31) INIT(2),
                05   MSG_RPC                  CHAR(04),
                05   MSG_RPC_L                FIXED BIN(31) INIT(4),
                05   MSG_TEXT                 CHAR(20),
                05   MSG_TEXT_L               FIXED BIN(31),
                05   MSG_SRVLIB,
                     10  MSG_SRVLIB_FUNC      CHAR(08) INIT(' '),
                     10  FILLER               CHAR(06) INIT(' RC = '),
                     10  MSG_SRVLIB_RC        PIC '----9';
DCL
             01  SWITCHES,
                 05  SEND_DONE                BIT(01) INIT('1'B),
                 05  SEND_DONE_ERROR          BIT(01) INIT('0'B),
                 05  SEND_DONE_OK             BIT(01) INIT('1'B),
                 05  TRACING                  BIT(01) INIT('0'B),
                 05  TRACING_ON               BIT(01) INIT('1'B);
/*-------------------------------------------------------------*/
  INITIALIZE_PROGRAM:
/*-------------------------------------------------------------*/
/*      -------------------------------------------------------*/
/*      establish gateway environment                          */
/*      -------------------------------------------------------*/
        CALL TDINIT (DFHEIBLK, GWL_RC, GWL_INIT_HANDLE);
/*      -------------------------------------------------------*/
/*      accept client request                                  */
/*      -------------------------------------------------------*/
        CALL TDACCEPT (GWL_PROC, GWL_RC, GWL_INIT_HANDLE,
                       SNA_CONNECTION_NAME,
                       SNA_SUBC);
/*      -------------------------------------------------------*/
/*      TDRESULT to validate kicked off via rpc request        */
/*      -------------------------------------------------------*/
        CALL TDRESULT (GWL_PROC, GWL_RC);
```

```
        IF GWL_RC NE TDS_PARM_PRESENT THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDRESULT';
            GO TO END_PROGRAM;
        END;

/*-------------------------------------------------------------*/
GET_PARM:
/*-------------------------------------------------------------*/
        CALL TDRCVPRM (GWL_PROC, GWL_RC,
                       GWL_RCVPRM_ID,
                       PARM_REQUEST,
                       TDSCHAR,
                       GWL_RCVPRM_MAX_DATA_L,
                       GWL_RCVPRM_DATA_L);

        IF PARM_REQUEST = PARM_REQUEST_INFACT THEN
            CALL TDINFACT_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_INFLOG THEN
            CALL TDINFLOG_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_LSTSPT THEN
            CALL TDLSTSPT_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETACT_ON THEN
            CALL TDSETACT_ON_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETACT_OFF THEN
            CALL TDSETACT_OFF_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETLOG_ON THEN
            CALL TDSETLOG_ON_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETLOG_OFF THEN
            CALL TDSETLOG_OFF_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETSPT_ON THEN
            CALL TDSETSPT_ON_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_SETSPT_OFF THEN
            CALL TDSETSPT_OFF_PROC;

        ELSE IF PARM_REQUEST = PARM_REQUEST_WRTLOG THEN
```

```
            CALL TDWRTLOG_PROC;

/*-------------------------------------------------------------*/
END_PROGRAM:
/*-------------------------------------------------------------*/
        IF SEND_DONE = SEND_DONE_OK THEN
            WRK_DONE_STATUS = TDS_DONE_COUNT;

        ELSE
        DO;
            CALL SRVLIB_ERROR;
            WRK_DONE_STATUS = TDS_DONE_ERROR;
            CTR_ROWS        = 0;
        END;

        CALL TDSNDDON (GWL_PROC, GWL_RC,
                       WRK_DONE_STATUS,
                       CTR_ROWS,
                       TDS_ZERO,
                       TDS_ENDRPC);

        CALL * (GWL_PROC, GWL_RC);
        EXEC CICS RETURN;

/*-------------------------------------------------------------*/
TDINFACT_PROC: PROC;
/*-------------------------------------------------------------*/
        WRKLEN1         = STG(GWL_INFACT_STATUS);
        WRKLEN2         = STG(CN_INFACT_STATUS);
        CTR_COLUMN      = CTR_COLUMN +1;
        MSG_SRVLIB_FUNC = 'TDESCRIB';

        CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN,
                       TDSINT4, WRKLEN1, GWL_INFACT_STATUS,
                       TDS_ZERO, TDS_FALSE, TDSINT4,
                       WRKLEN1, CN_INFACT_STATUS, WRKLEN2);


        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE = SEND_DONE_ERROR;
            GO TO TDINFACT_EXIT;
        END;

        WRKLEN1    = STG(GWL_INFACT_FILENAME);
        WRKLEN2    = STG(CN_INFACT_FILENAME);
```

```
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC,
               CTR_COLUMN,
               TDSCHAR,
               WRKLEN1,
               GWL_INFACT_FILENAME,
               TDS_ZERO,
               TDS_FALSE,
               TDSCHAR,
               WRKLEN1,
               CN_INFACT_FILENAME,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFACT_EXIT;
END;

WRKLEN1    = STG(GWL_INFACT_RECORDS);
WRKLEN2    = STG(CN_INFACT_RECORDS);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC,
               CTR_COLUMN,
               TDSINT4,
               WRKLEN1,
               GWL_INFACT_RECORDS,
               TDS_ZERO,
               TDS_FALSE,
               TDSINT4,
               WRKLEN1,
               CN_INFACT_RECORDS,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFACT_EXIT;
END;



CALL TDINFACT (GWL_INIT_HANDLE, GWL_RC,
               GWL_INFACT_STATUS,
```

```
                        GWL_INFACT_FILENAME,
                        GWL_INFACT_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFACT';
            GO TO TDINFACT_EXIT;
        END;

        CALL TDSNDROW (GWL_PROC, GWL_RC);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSNDROW';
            GO TO TDINFACT_EXIT;
        END;

        CTR_ROWS = CTR_ROWS +1;

/*-------------------------------------------------------------*/
TDINFACT_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDINFACT_PROC;


/*-------------------------------------------------------------*/
TDINFLOG_PROC: PROC;
/*-------------------------------------------------------------*/
        WRKLEN1         = STG(GWL_INFLOG_GLOBAL);
        WRKLEN2         = STG(CN_INFLOG_GLOBAL);
        CTR_COLUMN      = CTR_COLUMN +1;
        MSG_SRVLIB_FUNC = 'TDESCRIB';

        CALL TDESCRIB (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       TDSINT4,
                       WRKLEN1,
                       GWL_INFLOG_GLOBAL,
                       TDS_ZERO,
                       TDS_FALSE,
                       TDSINT4,
                       WRKLEN1,
```

```
                CN_INFLOG_GLOBAL,
                WRKLEN2);



IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_API);
WRKLEN2    = STG(CN_INFLOG_API);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC,
                CTR_COLUMN,
                TDSINT4,
                WRKLEN1,
                GWL_INFLOG_API,
                TDS_ZERO,
                TDS_FALSE,
                TDSINT4,
                WRKLEN1,
                CN_INFLOG_API,
                WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_HEADER);
WRKLEN2    = STG(CN_INFLOG_HEADER);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC,
                CTR_COLUMN,
                TDSINT4,
                WRKLEN1,
                GWL_INFLOG_HEADER,
                TDS_ZERO,
                TDS_FALSE,
                TDSINT4,
                WRKLEN1,
```

```
                CN_INFLOG_HEADER,
                WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_DATA);
WRKLEN2    = STG(CN_INFLOG_DATA);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN,
               TDSINT4, WRKLEN1,
               GWL_INFLOG_DATA,
               TDS_ZERO,
               TDS_FALSE,
               TDSINT4,
               WRKLEN1,
               CN_INFLOG_DATA,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_TRACEID);
WRKLEN2    = STG(CN_INFLOG_TRACEID);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN,
               TDSINT4, WRKLEN1,
               GWL_INFLOG_TRACEID,
               TDS_ZERO,
               TDS_FALSE,
               TDSINT4,
               WRKLEN1,
               CN_INFLOG_TRACEID,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
```

```
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_FILENAME);
WRKLEN2    = STG(CN_INFLOG_FILENAME);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC, CTR_COLUMN,
               TDSCHAR, WRKLEN1,
               GWL_INFLOG_FILENAME,
               TDS_ZERO,
               TDS_FALSE,
               TDSCHAR,
               WRKLEN1,
               CN_INFLOG_FILENAME,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

WRKLEN1    = STG(GWL_INFLOG_RECORDS);
WRKLEN2    = STG(CN_INFLOG_RECORDS);
CTR_COLUMN = CTR_COLUMN +1;

CALL TDESCRIB (GWL_PROC, GWL_RC,
               CTR_COLUMN,
               TDSINT4,
               WRKLEN1,
               GWL_INFLOG_RECORDS,
               TDS_ZERO,
               TDS_FALSE,
               TDSINT4,
               WRKLEN1,
               CN_INFLOG_RECORDS,
               WRKLEN2);

IF GWL_RC NE TDS_OK THEN
DO;
    SEND_DONE = SEND_DONE_ERROR;
    GO TO TDINFLOG_EXIT;
END;

CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
```

```
                        GWL_INFLOG_GLOBAL,
                        GWL_INFLOG_API,
                        GWL_INFLOG_HEADER,
                        GWL_INFLOG_DATA,
                        GWL_INFLOG_TRACEID,
                        GWL_INFLOG_FILENAME,
                        GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDINFLOG_EXIT;
        END;

        CALL TDSNDROW (GWL_PROC, GWL_RC);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSNDROW';
            GO TO TDINFLOG_EXIT;
        END;

        CTR_ROWS = CTR_ROWS +1;

/*-------------------------------------------------------------*/
TDINFLOG_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDINFLOG_PROC;

/*-------------------------------------------------------------*/
TDLSTSPT_PROC: PROC;
/*-------------------------------------------------------------*/
        WRKLEN1         = STG(WRK_TRANID);
        WRKLEN2         = STG(CN_LSTSPT_TRANID);
        CTR_COLUMN      = CTR_COLUMN +1;
        MSG_SRVLIB_FUNC = 'TDESCRIB';

        CALL TDESCRIB (GWL_PROC, GWL_RC,
                       CTR_COLUMN,
                       TDSCHAR,
                       WRKLEN1,
                       WRK_TRANID,
```

```
                        TDS_ZERO,
                        TDS_FALSE,
                        TDSCHAR,
                        WRKLEN1,
                        CN_LSTSPT_TRANID,
                        WRKLEN2);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE = SEND_DONE_ERROR;
            GO TO TDLSTSPT_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      find global status                                    */
/*      ------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                        GWL_INFLOG_GLOBAL,
                        GWL_INFLOG_API,
                        GWL_INFLOG_HEADER,
                        GWL_INFLOG_DATA,
                        GWL_INFLOG_TRACEID,
                        GWL_INFLOG_FILENAME,
                        GWL_INFLOG_RECORDS);




/*      ------------------------------------------------------*/
/*      if any error, then assume tracing disabled            */
/*      ------------------------------------------------------*/
        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDLSTSPT_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      if not specific tracing, then return nothing          */
/*      ------------------------------------------------------*/
        IF GWL_INFLOG_GLOBAL NE TDS_TRACE_SPECIFIC_RPCS THEN
            GO TO TDLSTSPT_EXIT;

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

```
/*      return rows                                                 */
/*      ----------------------------------------------------------*/
        CALL TDLSTSPT (GWL_INIT_HANDLE, GWL_RC,
                       GWL_LSTSPT_LIST(1));

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDLSTSPT';
        END;

        DOROW: DO WRK_LSTSPT_SS = 1 TO 8;
            WRK_TRANID = GWL_LSTSPT_LIST(WRK_LSTSPT_SS);

            CALL TDSNDROW (GWL_PROC, GWL_RC);

            IF GWL_RC NE TDS_OK THEN
            DO;
                SEND_DONE       = SEND_DONE_ERROR;
                MSG_SRVLIB_FUNC = 'TDSNDROW';
                LEAVE DOROW;
            END;

            CTR_ROWS = CTR_ROWS +1;
        END;

/*-------------------------------------------------------------*/
TDLSTSPT_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDLSTSPT_PROC;



/*-------------------------------------------------------------*/
TDSETACT_ON_PROC: PROC;
/*-------------------------------------------------------------*/
        CALL TDINFACT (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFACT_STATUS,
                       GWL_INFACT_FILENAME,
                       GWL_INFACT_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
```

```
            MSG_SRVLIB_FUNC = 'TDINFACT';
            GO TO TDSETACT_ON_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      turn on host accounting                               */
/*      ------------------------------------------------------*/
        CALL TDSETACT (GWL_INIT_HANDLE, GWL_RC,
                       TDS_TRUE,
                       GWL_INFACT_FILENAME,
                       GWL_INFACT_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETACT';
            GO TO TDSETACT_ON_EXIT;
        END;

/*------------------------------------------------------------*/
TDSETACT_ON_EXIT:
/*------------------------------------------------------------*/
        RETURN;

END TDSETACT_ON_PROC;

/*------------------------------------------------------------*/
TDSETACT_OFF_PROC: PROC;
/*------------------------------------------------------------*/
        CALL TDINFACT (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFACT_STATUS,
                       GWL_INFACT_FILENAME,
                       GWL_INFACT_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFACT';
            GO TO TDSETACT_OFF_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      turn off host accounting if on                        */
/*      ------------------------------------------------------*/
        IF GWL_INFACT_STATUS = TDS_TRUE THEN
        DO;
```

```
            CALL TDSETACT (GWL_INIT_HANDLE, GWL_RC,
                           TDS_FALSE,
                           GWL_INFACT_FILENAME,
                           GWL_INFACT_RECORDS);

            IF GWL_RC NE TDS_OK THEN
            DO;
                SEND_DONE       = SEND_DONE_ERROR;
                MSG_SRVLIB_FUNC = 'TDSETACT';
                GO TO TDSETACT_OFF_EXIT;
            END;
        END;

/*-------------------------------------------------------------*/
TDSETACT_OFF_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDSETACT_OFF_PROC;

/*-------------------------------------------------------------*/
TDSETLOG_ON_PROC: PROC;
/*-------------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC, GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API, GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA, GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME, GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDSETLOG_ON_EXIT;
        END;


/*      ------------------------------------------------------*/
/*      turn on API (CICS Aux Trace) and Header tracing       */
/*      ------------------------------------------------------*/
        CALL TDSETLOG (GWL_INIT_HANDLE, GWL_RC,
                       TDS_TRACE_ALL_RPCS, TDS_TRUE,
                       TDS_TRUE, GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);
```

```
        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETLOG';
            GO TO TDSETLOG_ON_EXIT;
        END;

/*--------------------------------------------------------------*/
TDSETLOG_ON_EXIT:
/*--------------------------------------------------------------*/
        RETURN;

END TDSETLOG_ON_PROC;

/*--------------------------------------------------------------*/
TDSETLOG_OFF_PROC: PROC;
/*--------------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDSETLOG_OFF_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      turn off API (CICS Aux Trace) and Header tracing      */
/*      ------------------------------------------------------*/
        CALL TDSETLOG (GWL_INIT_HANDLE, GWL_RC,
                       TDS_NO_TRACING,
                       TDS_FALSE,
                       TDS_FALSE,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);
```

```
        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETLOG';
            GO TO TDSETLOG_OFF_EXIT;
        END;

/*--------------------------------------------------------------*/
TDSETLOG_OFF_EXIT:
/*--------------------------------------------------------------*/
        RETURN;

END TDSETLOG_OFF_PROC;

/*--------------------------------------------------------------*/
TDSETSPT_ON_PROC: PROC;
/*--------------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDSETSPT_ON_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      turn on tracing for specific transactions             */
/*      ------------------------------------------------------*/
        CALL TDSETLOG (GWL_INIT_HANDLE, GWL_RC,
                       TDS_TRACE_SPECIFIC_RPCS,
                       TDS_TRUE,
                       TDS_TRUE,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
```

```
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETLOG';
            GO TO TDSETSPT_ON_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      enable error log recording for this tranid            */
/*      ------------------------------------------------------*/
        GWL_SETSPT_OPTIONS = 2;
        WRKLEN1            = STG(WRK_RPC);


        CALL TDSETSPT (GWL_INIT_HANDLE, GWL_RC,
                       TDS_TRUE,
                       GWL_SETSPT_OPTIONS,
                       WRK_RPC,
                       WRKLEN1);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETSPT';
            GO TO TDSETSPT_ON_EXIT;
        END;

/*--------------------------------------------------------------*/
TDSETSPT_ON_EXIT:
/*--------------------------------------------------------------*/
        RETURN;

END TDSETSPT_ON_PROC;

/*--------------------------------------------------------------*/
TDSETSPT_OFF_PROC: PROC;
/*--------------------------------------------------------------*/
/*      ------------------------------------------------------*/
/*      Assume specific tracing is on for this transaction,   */
/*      and turn it off.                                      */
/*      ------------------------------------------------------*/
        WRKLEN1 = STG(WRK_RPC);
        CALL TDSETSPT (GWL_INIT_HANDLE, GWL_RC,
                       TDS_FALSE,
                       GWL_SETSPT_OPTIONS,
                       WRK_RPC,
                       WRKLEN1);
```

```
        IF GWL_RC NE TDS_OK
            AND GWL_RC NE TDS_ENTRY_NOT_FOUND THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDSETSPT';
            GO TO TDSETSPT_OFF_EXIT;
        END;

/*-------------------------------------------------------------*/
TDSETSPT_OFF_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDSETSPT_OFF_PROC;

/*-------------------------------------------------------------*/
TDWRTLOG_PROC: PROC;
/*-------------------------------------------------------------*/
/*      -------------------------------------------------------*/
/*      write a log entry only if logging is enabled          */
/*      -------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

        IF GWL_RC NE TDS_OK THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO TDWRTLOG_EXIT;
        END;

/*      -------------------------------------------------------*/
/*      write a log entry only if logging is enabled          */
/*      -------------------------------------------------------*/
        CALL GET_TRACE_STATUS;

        IF (TRACING) THEN
        DO;
            CALL TDWRTLOG (GWL_PROC, GWL_RC,
```

```
                            TDS_TRUE,
                            GWL_WRTLOG_MSG,
                            GWL_WRTLOG_MSG_L);

            IF GWL_RC NE TDS_OK THEN
            DO;
                SEND_DONE       = SEND_DONE_ERROR;
                MSG_SRVLIB_FUNC = 'TDWRTLOG';
                GO TO TDWRTLOG_EXIT;
            END;
        END;

        ELSE
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'LOGNOTON';
            GO TO TDWRTLOG_EXIT;
        END;

/*-------------------------------------------------------------*/
TDWRTLOG_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END TDWRTLOG_PROC;

/*-------------------------------------------------------------*/
GET_TRACE_STATUS: PROC;
/*-------------------------------------------------------------*/
/*      -------------------------------------------------------*/
/*      find global status                                     */
/*      -------------------------------------------------------*/
        CALL TDINFLOG (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFLOG_GLOBAL,
                       GWL_INFLOG_API,
                       GWL_INFLOG_HEADER,
                       GWL_INFLOG_DATA,
                       GWL_INFLOG_TRACEID,
                       GWL_INFLOG_FILENAME,
                       GWL_INFLOG_RECORDS);

/*      -------------------------------------------------------*/
/*      if any error, then assume tracing disabled             */
/*      -------------------------------------------------------*/
        IF GWL_RC NE TDS_OK THEN
        DO;
```

```
            SEND_DONE       = SEND_DONE_ERROR;
            MSG_SRVLIB_FUNC = 'TDINFLOG';
            GO TO GET_TRACE_STATUS_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      if global tracing on, then tracing enabled            */
/*      ------------------------------------------------------*/
        IF GWL_INFLOG_GLOBAL = TDS_TRACE_ALL_RPCS THEN
        DO;
            TRACING = TRACING_ON;
            GO TO GET_TRACE_STATUS_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      if error logging on, then tracing enabled             */
/*      ------------------------------------------------------*/
        IF GWL_INFLOG_GLOBAL = TDS_TRACE_ERRORS_ONLY THEN
        DO;

            TRACING = TRACING_ON;
            GO TO GET_TRACE_STATUS_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      if specific tracing not on, then no tracing on        */
/*      ------------------------------------------------------*/
        IF GWL_INFLOG_GLOBAL NE TDS_TRACE_SPECIFIC_RPCS THEN
        DO;
            GO TO GET_TRACE_STATUS_EXIT;
        END;

/*      ------------------------------------------------------*/
/*      specific tracing is on, see if on for this transaction*/
/*      ------------------------------------------------------*/
        WRKLEN1 = STG(WRK_RPC);

        CALL TDINFSPT (GWL_INIT_HANDLE, GWL_RC,
                       GWL_INFSPT_STATUS,
                       GWL_INFSPT_OPTIONS,
                       WRK_RPC,
                       WRKLEN1);

        IF (GWL_RC NE TDS_OK AND GWL_RC NE TDS_ENTRY_NOT_FOUND) THEN
        DO;
            SEND_DONE       = SEND_DONE_ERROR;
```

```
            MSG_SRVLIB_FUNC = 'TDINSPT';
            GO TO GET_TRACE_STATUS_EXIT;
        END;

        IF GWL_INFSPT_STATUS = TDS_TRUE THEN
        DO;
            TRACING = TRACING_ON;
        END;

/*-------------------------------------------------------------*/
GET_TRACE_STATUS_EXIT:
/*-------------------------------------------------------------*/
        RETURN;

END GET_TRACE_STATUS;

/*-------------------------------------------------------------*/
SRVLIB_ERROR: PROC;
/*-------------------------------------------------------------*/
        MSG_SRVLIB_RC = GWL_RC;
        MSG_TEXT      = STRING(MSG_SRVLIB);
        MSG_TEXT_L    = STG(MSG_SRVLIB);
        MSG_TYPE      = TDS_ERROR_MSG;
        MSG_RPC       = WRK_RPC;


 /*-------------------------------------------------------------------*/
SEND_ERROR_MESSAGE: PROC;
/*-------------------------------------------------------------------*/
        SEND_DONE    = SEND_DONE_ERROR;
        MSG_SEVERITY = MSG_SEVERITY_ERROR;
        MSG_NR       = MSG_NR_ERROR;
        MSG_TYPE     = TDS_ERROR_MSG;
        CALL SEND_MESSAGE;

        RETURN;

END SEND_ERROR_MESSAGE;

/*-------------------------------------------------------------------*/
SEND_MESSAGE: PROC;
/*-------------------------------------------------------------------*/

/*      -------------------------------------------------------------*/
/*      ensure we're in right state to send a message                */
/*      -------------------------------------------------------------*/
```

```
        CALL TDSTATUS (GWL_PROC, GWL_RC,
                       GWL_STATUS_NR,
                       GWL_STATUS_DONE,
                       GWL_STATUS_COUNT,
                       GWL_STATUS_COMM,
                       GWL_STATUS_RETURN_CODE,
                       GWL_STATUS_SUBCODE);

        IF GWL_RC = TDS_OK THEN
        DO;
            IF GWL_STATUS_COMM = TDS_RECEIVE THEN

            CALL TDSNDMSG (GWL_PROC, GWL_RC,
                           MSG_TYPE, MSG_NR,
                           MSG_SEVERITY,
                           TDS_ZERO,
                           TDS_ZERO,
                           MSG_RPC, MSG_RPC_L,
                           MSG_TEXT, MSG_TEXT_L);
        END;

        RETURN;

END SEND_MESSAGE;

        RETURN;

END SRVLIB_ERROR;

END SYSAMP1;
```

Index

A

C

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

V